Titre: Title:	Représentation abstraite pour l'animation automatique de scènes virtuelles par planification classique
Auteur: Author:	Jonathan Tardif
Date:	2012
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Tardif, J. (2012). Représentation abstraite pour l'animation automatique de scènes virtuelles par planification classique [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/992/

UNIVERSITÉ DE MONTRÉAL

REPRÉSENTATION ABSTRAITE POUR L'ANIMATION AUTOMATIQUE DE SCÈNES VIRTUELLES PAR PLANIFICATION CLASSIQUE

JONATHAN TARDIF
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2012

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

REPRÉSENTATION ABSTRAITE POUR L'ANIMATION AUTOMATIQUE DE SCÈNES VIRTUELLES PAR PLANIFICATION CLASSIQUE

présenté par : TARDIF Jonathan
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury d'examen constitué de :

M. DAGENAIS Michel, Ph.D., président.
M. OZELL Benoît, Ph.D., membre et directeur de recherche.
M. GAGNON Michel, Ph.D., membre et codirecteur de recherche.
Mme ZOUAQ Amal, Ph.D., membre.

REMERCIEMENTS

J'aimerais d'abord remercier mes directeurs de recherche Michel Gagnon et Benoît Ozell pour cette opportunité de recherche, le partage de leur expertise et leur encadrement tout au long de mon parcours en tant qu'étudiant à la maîtrise. J'aimerais également remercier le Fonds québécois de la recherche sur la nature et les technologies (FQRNT) ainsi que la Fondation de Polytechnique pour leur support financier m'ayant permis d'entreprendre et de compléter mes études supérieures. De plus, j'aimerais remercier mes collègues étudiants Guillaume Bouilly, Paul Gédéon, François-Xavier Desmarais et Maxime Ouellet de m'avoir accompagné dans ce projet et d'avoir rendu cette expérience amusante au quotidien. Enfin, j'aimerais remercier ma famille et mes proches pour leur support inconditionnel et leurs encouragements tout au long de mon cheminement d'études, sans quoi je n'aurais pu traverser cette période sans heurts.

RÉSUMÉ

La génération d'animations dans des scènes 3D nécessite un travail manuel important de la part des créateurs. Pour cette raison, nous tentons d'obtenir une certaine automatisation du processus d'animation d'une scène virtuelle. Parmi les diverses techniques possibles, nous nous intéressons à la planification classique, dont le but est de générer un plan à partir d'un problème donné en formulant les buts spécifiques à atteindre.

Notre projet s'inscrit dans les travaux du projet GITAN, dont l'objectif global est la génération automatique d'animations 3D à partir de texte. Le but de notre projet est de démontrer la faisabilité de la génération automatique d'un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite.

Notre solution est d'abord basée sur la définition d'un format de description de scène inspiré de concepts définis par l'ontologie *GUM-Space* ainsi qu'une représentation des actions basée sur la ressource sémantique *FrameNet*. Selon le type d'action, plusieurs attributs peuvent qualifier le déroulement de celle-ci en fonction des objets concernés où encore selon des modalités spatiales.

Afin d'ajouter l'information nécessaire à la formulation d'un problème de planification, nous avons conçu et développé une ontologie d'objets du monde réel et de contrôleurs en langage OWL. Cette dernière permet d'enrichir chaque situation décrite par de l'information concrète sur les fonctionnalités offertes par les objets de la scène. Elle fournit également de l'information utile au déroulement de l'action dans la scène. Cette base de connaissances est combinée à la définition d'un domaine d'actions en langage PDDL pour la résolution du problème de planification formulé à partir de la représentation conceptuelle abstraite.

Nous avons mis au point une méthodologie d'évaluation impliquant l'implémentation de 16 scénarios de tests divers que nous avons validés manuellement afin de vérifier la validité de notre solution. Nous avons ensuite procédé à une analyse de la couverture du domaine d'actions du monde réel. Cette dernière a révélé des taux de couverture potentielle de 50% à 100% pour chacune des catégories d'actions auxquelles nous nous intéressions, avec un taux global de 88%.

Au terme de l'analyse des résultats de notre expérimentation, nous sommes en mesure d'affirmer qu'il est possible de générer automatiquement un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite. Nous affirmons cela sous certaines limitations posées par notre solution et notre méthodologie d'évaluation, mais nous pouvons tout de même affirmer que nous avons atteint nos objectifs de recherche.

ABSTRACT

Creation of 3D animations in virtual environments implies a lot of manual work from the hands of creators. Considering that, we aim to automate a part of this animation process by including an automated planning approach. The purpose of such an approach is to generate a plan from a given problem by formulating specific goals to reach.

Our project is included in the GITAN project, whose goal is to generate 3D animations automatically from textual inputs. The specific goal of our project is to demonstrate the feasibility of animation plan generation from a conceptual representation offering a high level of abstraction, while using an automated planning approach.

The solution we suggest involves defining a scene description format inspired from classes defined by the *GUM-Space* ontology. The resource used for the classification of actions is *FrameNet*, from which we can add parameters to describe the action's progress such as object-using constraints or spatial modalities.

In order to add the required information for the formulation of a planning problem, we developed an OWL ontology to represent real-world objects and controllers. Using this ontology allows us to enhance each scene description by adding information on objects' features and action's progress. To get along with this knowledge base, we also defined an action domain in PDDL language for resolving the formulated planning problem.

Our evaluation methodology involved the implementation of 16 test cases, from which we validated the results manually in order to confirm the validity of our solution. We then conducted an analysis of the action domain's potential coverage, which revealed coverage rates from 50% to 100% for the categories that we focused on, with a global coverage rate of 88%.

After our results' analysis, we can confirm the feasibility of our automated planning approach for animation plan generation from a conceptual representation. We consider a few limitations to our solution and evaluation methodology that are restricting our work's scope, but we can still assert that we have reached our research objectives.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SIGLES ET ABRÉVIATIONS

3D	Trois Dimensions
A-BOX	Assertion Box
ADL	Action Definition Language
API	Application Programming Interface
CSP	Constraint Satisfaction Problem
GITAN	Grammaire pour l'Interprétation de Textes et d'ANimations
GUM	Generalized Upper Model
HLA	High-Level Action
HTN	Hierarchical Task Networks
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IRI	Internationalized Resource Identifier
OWL	Web Ontology Language
PDDL	Planning Domain Definition Language
POP	Partial-Order Planning
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SAT	Satisfiability
SPARQL	SPARQL Protocol And RDF Query Language
STRIPS	STanford Research Institute Problem Solver
SUMO	Suggested Upper Merged Ontology
SUO	Standard Upper Ontology
T-BOX	Terminology Box
TAL	Traitement Automatique de la Langue
XML	Extensible Markup Language
XSD	XML Schema Definition

CHAPITRE 1

INTRODUCTION

Depuis maintenant plus d'un siècle, l'être humain s'intéresse à l'animation d'images pour la représentation d'éléments en action. Initialement composées d'images en défilement rapide, les animations sont maintenant des environnements complexes en trois dimensions dont un ou plusieurs éléments sont en mouvement. Elles se retrouvent aujourd'hui dans plusieurs facettes de notre vie, par exemple le cinéma, la télévision, les jeux vidéo ainsi que tout autre contenu multimédia qui nous est accessible.

La génération de ces animations nécessite un travail manuel important de la part des créateurs. Pour cette raison, nous tentons d'obtenir une certaine automatisation du processus d'animation d'une scène virtuelle. Parmi les diverses techniques possibles, nous nous intéressons à la planification classique. Le but de cette technique est de générer un plan à partir d'un problème donné en formulant les buts spécifiques à atteindre.

La planification classique est une méthode éprouvée dans le domaine de la résolution de problèmes ayant trait à l'exécution d'une séquence d'actions. Nous croyons qu'il est possible d'appliquer cette technique pour la génération automatique d'une animation dans une scène virtuelle. Le défi consiste en la formulation d'un problème résoluble par planification à partir d'informations abstraites décrivant l'animation.

Le but de notre projet de recherche est donc de permettre la génération d'un plan d'animation dans un contexte de scène virtuelle à partir d'informations abstraites décrivant une situation à animer. Les orientations de notre projet concernent principalement la représentation initiale de la scène animée ainsi que l'ajout de connaissances dans notre environnement virtuel.

Afin de parvenir à ce but, nous proposons un format de représentation conceptuelle permettant de décrire la scène à un haut niveau d'abstraction. Nous proposons également un format de représentation des connaissances permettant l'ajout de l'information nécessaire à la formulation d'un problème de planification.

Notre projet de recherche s'inscrit dans le cadre global du projet GITAN, signifiant Grammaire pour l'Interprétation de Textes et d'ANimations. Le but du projet GITAN est de permettre la génération automatique d'animations à partir de textes. Pour ce faire, plusieurs phases de traitement de l'information sont définies, telles que l'analyse syntaxique et sémantique du texte, la construction d'une scène virtuelle en trois dimensions et la génération de l'animation dans la scène virtuelle. Notre projet est inclus dans la phase de génération de

l'animation, où il constitue la première étape du traitement. Ainsi, nous supposons que l'information du texte a été analysée avec succès pour pouvoir être représentée dans un format conceptuel abstrait. Nous supposons également qu'une scène virtuelle a été générée à partir des informations obtenues de l'analyse de texte. Nous visons à définir ce format de représentation conceptuelle abstraite de la scène afin de permettre la génération du plan d'animation. Ce dernier doit permettre l'exécution de l'action décrite dans la situation initiale tout en respectant les contraintes posées par la scène statique fournie.

D'abord, le chapitre 2 présente en détails la problématique associée à notre projet de recherche. Celle-ci inclut la présentation de la solution envisagée ainsi que de notre question de recherche, nos hypothèses de recherche et nos objectifs de recherche.

Ensuite, les chapitres 3 et 4 constituent une revue de littérature concernant les principaux sujets abordés par notre projet de recherche. Le chapitre 3 présente d'abord un état de l'art ayant trait aux bases de connaissances, à la sémantique dans les environnements virtuels ainsi qu'à la conceptualisation du monde réel dans des ressources sémantiques. Puis, le chapitre 4 présente la planification classique ainsi que les différentes approches qui y sont associées pour la formulation et la résolution de problèmes.

La solution proposée afin de résoudre la problématique est présentée en entier au chapitre 5. Nous y présentons les principaux aspects de notre solution, incluant les différents formats de représentation de l'information que nous avons définis ainsi que le processus de formulation d'un problème de planification qui en découle.

Nous définissons ensuite une méthodologie d'évaluation de notre solution présentée au chapitre 6. Ce dernier présente les détails de développement du programme et de configuration des outils nécessaires à la réalisation de notre expérience, pour ensuite décrire cette dernière dans son ensemble, incluant les scénarios de test ainsi que les mécanismes d'évaluation des résultats.

Le chapitre 7 présente les résultats obtenus à la suite de notre expérience. Nous proposons ensuite une analyse de ceux-ci suivie d'une discussion associée à cette analyse. Cette discussion présente l'impact de plusieurs aspects de notre solution envers l'atteinte de nos résultats ainsi que les limitations associées à notre méthodologie d'évaluation.

Enfin, le chapitre 8 présente brièvement les orientations futures des travaux visant à améliorer notre solution. Nous évaluons également les travaux futurs à entreprendre afin de compléter l'intégration de nos travaux dans le projet GITAN.

CHAPITRE 2

PROBLÉMATIQUE

Ce chapitre a pour but de présenter la problématique associée à l'animation automatique de scènes virtuelles par planification. Nous exposons d'abord la situation actuelle concernant la représentation de scènes et l'animation de celles-ci, puis nous présentons les différentes phases du projet qui permettront d'atteindre un objectif principal. Suite à l'exposition de cette problématique, nous présentons le but de notre projet, la question de recherche ainsi que nos hypothèses et objectifs de recherche.

2.1 Situation actuelle

Depuis les débuts de la modélisation de scènes virtuelles, le processus d'animation de celles-ci s'est largement complexifié. Ceci est d'autant plus vrai depuis les avancements graphiques impressionnants effectués au cours des dernières années sur le plan des performances des cartes graphiques. Le niveau de détail graphique grandissant permis grâce à ces avancements a pour conséquence que l'édition de scènes graphiques pour l'animation de scènes 3D demande un temps de création de plus en plus important (Tutenel *et al.* (2008)).

Ce processus d'animation comprend plusieurs étapes. Tout d'abord, un certain travail de création de la scène doit être fait. Ce travail est indépendant du processus d'animation de la scène, mais est bien sûr essentiel à celui-ci. Les composantes de la scène doivent donc d'abord être modélisées en 3D, souvent à l'aide d'un logiciel éditeur. Ensuite, les mouvements internes et externes de ces objets doivent être programmés dans le cas d'une scène dynamique, c'est-à-dire une scène dont le contenu est modifié pendant l'exécution de l'animation. Puis, ils doivent être disposés correctement afin de permettre une interactivité lors de l'animation, ainsi qu'une certaine cohérence à des fins de compréhension par un être humain.

Une fois le travail de création de la scène effectué, nous pouvons nous pencher vers le processus d'animation. Ce dernier demande à une personne de d'abord interpréter correctement la scène à animer, c'est-à-dire de bien comprendre ce que la scène animée doit représenter. Un travail minutieux de positionnement des éléments de la scène en fonction du temps s'ensuit. Ce travail doit tenir en compte l'interaction des acteurs de l'animation avec les objets positionnés dans la scène, la modification de la position des acteurs et des objets impliqués en fonction du temps, de même que les modifications à la structure interne d'un objet en mouvement (un corps humain par exemple). Évidemment, puisque la scène varie en fonction

du temps, l'état courant de celle-ci à un temps donné doit toujours est pris en compte afin de toujours conserver les aspects d'interactivité et de cohérence énoncés précédemment.

2.2 Défis

Le contexte du projet GITAN amène à une automatisation la plus complète possible du processus d'animation d'une scène virtuelle. La partie à laquelle nous nous intéressons est la phase « planification » de l'animation, c'est-à-dire l'interprétation d'une certaine quantité d'information sur la scène décrite pour la génération d'un plan décrivant le déroulement de l'animation. Le point central de l'information véhiculée par une scène animée est l'action principale se déroulant dans celle-ci. Afin de pouvoir représenter concrètement cette information, il est important que cette action soit de type mouvement dans l'espace et que les interactions qu'elle implique avec les objets de la scène ainsi que leurs effets soient visiblement perceptibles. Nous pouvons ensuite démontrer la faisabilité d'une solution de grande envergure en définissant un environnement connu et borné dans lequel peut être développé une solution adéquate, pour ensuite démontrer son applicabilité au reste du monde. Dans notre cas, nous avons choisi un environnement virtuel à l'échelle humaine composé d'objets utilisés à l'intérieur d'une maison et d'actions pouvant être effectuées par un acteur humain.

La phase de planification de l'animation comporte comme contrainte préalable la génération d'une scène statique à partir des mêmes informations de départ, permettant ainsi la génération d'une animation dans celle-ci. Le travail de génération d'un plan d'animation comporte une phase de traitement de l'information obtenue en entrée décrivant la situation à animer, une phase d'analyse du contenu de la scène statique créée en parallèle, une phase d'utilisation de connaissance de sens commun ainsi qu'une phase de génération d'un plan à partir des informations obtenues dans les trois premières phases.

Premièrement, la phase de traitement de l'information obtenue en entrée est une tâche difficile car il s'agit d'information provenant initialement de texte. Le défi est de trouver un format de représentation permettant à la fois la conservation de la plus grande quantité d'information préalablement obtenue tout en permettant un traitement efficace lors de l'interprétation. De plus, ce format doit être apte à représenter des situations statiques puisqu'il sert également de point de départ à la génération de la scéne statique. Pour les fins d'animation, ce format doit d'abord intégrer une forme de conceptualisation des actions décrites dans le texte analysé au départ. De plus, les acteurs, objets et autres composantes physiques de la scène doivent également être représentés de façon conceptuelle, afin d'éviter les ambiguïtés et aussi afin de pouvoir obtenir de l'information supplémentaire lors du traitement via une base de connaissances. Le format doit également permettre d'exprimer le rôle de ces acteurs

et objets dans la scène animée, et bien sûr de représenter l'information spatiale présente initialement dans le texte, sans quoi l'animation ne peut y être fidèle.

Deuxièmement, la phase d'analyse du contenu de la scène statique créée en parallèle est essentielle à la dernière phase de planification. La génération de cette scène statique est le sujet d'un projet parallèle et inclus dans le projet GITAN. Elle a pour but d'instancier les objets nécessaires au déroulement de l'animation et de les positionner correctement dans la scène virtuelle. Sans pour autant connaître le contenu entier de la scène (coordonnées exactes, orientation des objets, etc.), les objets instanciés et leur position relative dans la scène sont de l'information cruciale à notre projet. Un format de représentation simple pour cette information doit être défini.

Troisièmement, la phase d'utilisation de connaissances de sens commun est un point crucial du projet. Toute l'information obtenue lors des deux premières phases doit être complétée par l'ajout de connaissance de sens commun. Cette connaissance, contenue dans une base de connaissances devant être trouvée ou créée, doit indiquer les aspects cruciaux des objets présents dans la scène par rapport à l'animation de celle-ci. L'aspect principal concernant l'animation est une hiérarchie de concepts axée sur la fonctionnalité (exemple : un verre est un contenant à liquide) permettant l'interaction des acteurs avec les objets. Par contre, comme cette base de connaissances est partagée avec le projet parallèle de génération de la scène, elle doit contenir des informations supplémentaires, telles que les objets nécessaires à la réalisation d'une action, par exemple, ainsi que quelques informations spatiales pour le positionnement des objets. Une seconde partie cruciale de la connaissance utilisée est le domaine d'actions disponibles pour la génération du plan d'action. Les actions possibles du domaine sont directement reliées à la fonctionnalité des objets, mais font aussi partie du processus de génération de plan.

Finalement, la phase de génération de plan consiste en la consolidation des informations recueillies lors des trois premières phases afin de les formaliser en un problème de planification. Ce problème, combiné au domaine d'actions, pourra être résolu par un planificateur classique et ainsi fournir un plan, consistant en une série ordonnée d'actions avec paramètres définis.

2.3 But

Considérant la problématique et les défis énoncés aux sections 2.1 et 2.2, le but de notre projet est d'être capable de générer un plan d'animation, c'est-à-dire une séquence ordonnée d'actions, dans une scène virtuelle enrichie sémantiquement à partir d'une description conceptuelle de la scène animée. Sachant que la planification classique est une méthode valide pour la résolution de ce type de problème (Russell et Norvig (2010)), nous visons à définir un

domaine d'actions basé sur des informations de sens commun et à transformer la description conceptuelle de la scène animée en un problème concret de planification.

Considérant les contraintes supplémentaires causées par l'inclusion de notre projet dans le projet GITAN, notre but est que le format de représentation conceptuelle de la scène animée offre un niveau d'abstraction suffisant pour permettre que l'analyse initiale d'un texte puisse s'y convertir. Le manque d'information concrète relié à ce niveau d'abstraction doit être comblé par l'ajout de connaissances de sens commun adaptées au problème de planification classique, sur le plan des fonctionnalités des objets.

Dans le but d'évaluer le travail réalisé, nous considérons deux points principaux comme facteurs d'évaluation. En premier lieu, la faisabilité de générer un plan d'animation automatiquement doit être démontrée. Ensuite, un test de couverture potentielle doit être effectué afin de démontrer l'applicabilité de notre solution à une partie suffisante du domaine de conceptualisation des actions.

Suite à l'expression de la situation actuelle, de la problématique et du but du projet, nous formulons la question de recherche suivante :

> Est-il possible de générer automatiquement un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite ?

2.4 Hypothèses de recherche

L'hypothèse principale de notre projet répond directement à la question de recherche. Nous posons l'hypothèse qu'il est possible de générer automatiquement un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite. Cette automatisation consiste à générer une séquence d'actions complète et représentative de la situation exprimée sous forme conceptuelle abstraite, tout en étant cohérente avec les informations de rôles et spatiales présentes dans celle-ci.

Nous décomposons cette hypothèse principale pour obtenir les hypothèses de recherche suivantes :

H1 Il est possible de concevoir un format de représentation conceptuelle abstraite contenant les informations essentielles à la création d'une scène animée.

H2 Il est possible d'obtenir l'information nécessaire à la formulation d'un problème de planification à partir d'une base de connaissances.

H3 Il est possible de formuler un problème de planification classique résoluble à partir de la représentation conceptuelle abstraite de la scène animée.

2.5 Objectifs de recherche

L'objectif principal de notre projet vise à évaluer notre hypothèse de recherche, c'est-à-dire vérifier la faisabilité de la génération automatique d'un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite.

Afin d'y parvenir, nous avons défini plusieurs objectifs de recherche spécifiques :

O1 Identifier les informations essentielles à la création d'une scène animée.

O2 Identifier un format de représentation conceptuelle abstraite de la scène animée.

O3 Identifier un format de représentation de l'information contenue dans la scène statique.

O4 Identifier un format de représentation des connaissances axé sur la fonctionnalité des éléments. Ce format doit être compatible avec le projet parallèle de génération de scène statique.

O5 Concevoir un programme capable de traiter l'information provenant des diverses sources pour en formuler un problème de planification classique.

O6 Vérifier la faisabilité de la génération d'un plan d'animation à partir d'une représentation conceptuelle abstraite.

O7 Démontrer l'applicabilité de notre solution à une partie suffisante du domaine de conceptualisation des actions.

CHAPITRE 3

REPRÉSENTATION SÉMANTIQUE DES ENVIRONNEMENTS VIRTUELS

En informatique, la représentation des connaissances consiste en l'organisation, le stockage et l'utilisation d'informations dans un cadre défini. Ce chapitre a pour but de présenter le concept de bases de connaissances comme outil pour la représentation d'information sémantique, ainsi que les différentes approches d'intégration de cette information dans les environnements virtuels et les banques de données de conceptualisation du monde réel.

3.1 Bases de connaissances

Une base de connaissances est en premier lieu un mécanisme de représentation de l'information relative à un monde donné. Elle consiste en un ensemble d'*énoncés* exprimés dans un langage précis, nommé *langage de représentation des connaissances*, qui représentent des assertions relatives au domaine concerné (Russell et Norvig (2010)). La construction d'une base de connaissances se fait par l'ajout un à un d'*axiomes*, qui représentent des énoncés donnés (en opposition aux énoncés inférés). Pour que la base de connaissances soit cohérente, ces axiomes ne doivent pas se contredire, c'est-à-dire qu'une classe d'individus ne doit pas être impossible à satisfaire, donc restreinte à l'ensemble vide.

L'interface standard d'une base de connaissances doit au minimum comporter des mécanismes permettant l'ajout de nouveaux énoncés (enrichissement de la base de connaissances) ainsi qu'une interface de requête, permettant d'obtenir l'information contenue dans la base. Ces mécanismes impliquent possiblement la notion d'*inférence logique*, c'est-à-dire l'ajout par déduction logique de nouveaux énoncés à partir de l'information déjà contenue dans la base. L'outil permettant la création de ces nouveaux énoncés à partir de raisonnement logique est nommé *raisonneur*. Il existe plusieurs types d'algorithmes d'inférence logique, mais tous sont basés sur deux propriétés essentielles : la validité des inférences et la complétude de l'algorithme (Russell et Norvig (2010)).

3.1.1 Formalisme logique

À la base de la représentation d'une base de connaissances se retrouve un formalisme logique. Ce dernier permet, pour un domaine déterminé, la définition de *concepts* et de *relations* entre les instances de ces derniers. L'ensemble de formalismes permettant de représenter la base de connaissances que nous utilisons dans ce projet est nommée *logique descriptive*.

Ces formalismes sont définis par une *syntaxe*, qui présente un certain nombre d'énoncés atomiques, d'opérateurs logiques ainsi que d'énoncés complexes, composés d'énoncés atomiques et d'opérateurs.

La *sémantique* d'un modèle logique consiste en la détermination de la vérité d'un énoncé dans son contexte. En logique descriptive, il s'agit de pouvoir déterminer la valeur de vérité (vrai ou faux) de tous les symboles en fonction du modèle donné (Russell et Norvig (2010)).

De manière plus concrète, l'ensemble des concepts et des énoncés représentant la sémantique d'un domaine d'information peut être regroupé dans une *ontologie*. Cette dernière consiste en la définition du modèle représentant un ensemble de concepts ainsi que les relations qui affectent ceux-ci. Les éléments principaux constituant une ontologie sont les suivants :

Classes Aussi nommés **Concepts**, ces éléments représentent la structure de base de l'ontologie. Combinées à des relations de type hiérarchique (comparable à la notion d'héritage en programmation orientée-objet), elles définissent une hiérarchie conceptuelle entre les différents types d'objets définis dans le domaine. Il existe deux types de classes : les classes *nommées* et les classes *anonymes*. Chaque classe nommée possède son identifiant unique et peut posséder un nombre illimité de *relations* avec d'autres classes et d'*attributs*. Par exemple, les classes *Humain*, *Homme* et *Femme* sont des classes nommées et les classes *Homme* et *Femme* sont des sous-classes de la classe *Humain*. Les classes anonymes correspondent à l'application d'une restriction sur une classe nommée. Par exemple, la classe *Femme* a comme classe de base la classe nommée *Humain*, mais aussi une restriction sur l'attribut *Sexe*, dont la valeur doit être 'F'. La classe *Femme* est donc aussi une sous-classe de la classe anonyme « Sexe = 'F' ».

Propriétés Ces éléments représentent les relations entre les différentes objets de l'ontologie. Chaque propriété définie peut posséder un *domaine* et une *image*, qui spécifient une restriction sur les classes d'individus pouvant être reliées par cette propriété. Par exemple, la propriété *aUneSoeur* aurait comme domaine la classe *Humain* et comme image la classe *Femme*.

Attributs Ces éléments représentent des caractéristiques que les objets peuvent posséder sous forme d'information *littérale*, c'est-à-dire représentable par un type standard de base (*int*, *float*, *char*, *string*, *date*, etc.). Contrairement aux propriétés, les attributs ne font pas de relations entre deux objets et ajoutent plutôt de l'information spécifique à l'objet. Par exemple, un individu de la classe *Humain* pourrait avoir un attribut *âge* de type *int*, auquel serait attribué le nombre entier correspondant à l'âge de la personne.

Individus Ces éléments sont simplement les *objets* qui composent la base de connaissances, c'est-à-dire qu'ils sont des instances des classes définies dans l'ontologie.

Le formalisme de logique descriptive divise l'information contenue dans la base de connaissances en deux parties distinctes : la *terminologie* (aussi nommée *T-BOX*) et les *assertions* (aussi nommées *A-BOX*). Premièrement, la terminologie correspond à la description du domaine. Elle comprend l'information globale de la base de connaissances, c'est-à-dire la définition des concepts, de leurs propriétés et de leurs restrictions. Dans une base de connaissances de type ontologie, cela comprend la définition explicite de ses classes, la déclaration des propriétés et de leurs paramètres (domaine, image, cardinalité, etc.) ainsi que la déclaration des attributs et de leur type. Cette information est générique et valable pour tous les individus du modèle. Deuxièmement, les assertions correspondent à la déclaration des individus peuplant le monde défini par la terminologie. Chaque individu de la A-BOX est soumis aux règles et restrictions énoncées dans la T-BOX, et il peuvent être assujettis à des propriétés et des attributs, toujours dans le respect de la terminologie. Dans une base de connaissances de type ontologie, chaque individu déclaré se voit attribuer au minimum une classe, est mis en relation avec les autres individus par des propriétés et peut comporter des attributs dont la valeur doit avoir été spécifiée.

3.1.2 Ontologies et web sémantique

Une fois qu'une ontologie a été définie (voir section 3.1.1), elle doit être traduite dans un format compréhensible autant par l'humain qui la crée et l'enrichit d'information que par la machine qui l'interroge et emmagasine l'information. Le format de base de représentation de la connaissance dans une ontologie est le modèle *RDF* (Resource Description Framework). Lassila et Swick (1998) ont présenté ce modèle et sa syntaxe afin de produire un langage universel permettant l'ajout d'information de type méta-données dans des base de données, c'est-à-dire de la connaissance supplémentaire concernant l'information dans la base. Une base de connaissances exprimée dans ce format - ainsi que son extension *RDF-Schema*, ou *RDFS* - est composée d'un ensemble de *triplets* (*rdf.Statement*) ajoutant chacun une information unique à la base. Ces triplets sont toujours de format « rdf.Resource rdf.Property rdf.Resource », où rdf.Resource correspond à une ressource quelconque de la base et rdf.Property englobe les propriétés et attributs (voir section 3.1.1).

Berners-Lee *et al.* (2001) ont utilisé la notion de bases de connaissances expliquée jusqu'à présent pour l'étendre à un éventuel partage de données de grande envergure qu'ils ont nommé le *web sémantique*. Ce dernier naît du principe que le web actuel est rempli d'informations, mais que celles-ci ne sont aucunement schématisées dans un format standard ni compréhensible par une machine. Le projet du web sémantique reprend donc la notion de bases de connaissances et tente de l'appliquer à la plus grande quantité possible d'informations déjà présentes sur le web actuel, afin de faciliter le partage de la connaissance, son

traitement ainsi qu'une structuration de ces informations.

Le langage le plus commun pour la représentation des ontologies sur le web est le *Web Ontology Langage* (*OWL*). McGuinness et van Harmelen (2004) ont présenté ce langage comme une extension au modèle RDF permettant une meilleure intégration des ontologies au web, tout en incluant de nouveaux concepts logiques ainsi qu'un amélioration des mécanismes d'inférence. De plus, Prud'hommeaux et Seaborne (2008) ont présenté le langage de requêtes *SPARQL* (*SPARQL Protocol and RDF Query Language*) permettant la recherche et même l'ajout ou la modification de données sur le web sémantique, ou dans une ontologie isolée.

Avant l'arrivée des ontologies partagées sur le web sémantique, certains projets ont oeuvré dans la même optique de partage de connaissances. Lenat (1995) a présenté la base de connaissances *CYC*, un projet ayant pour but de regrouper à un seul endroit un très grand nombre d'informations de sens commun (*common sense knowledge*). Ces informations sont très utiles pour des programmes de traitement automatique de la langue (TAL) ainsi que d'apprentissage automatique. Un projet d'ontologie libre nommé *OpenCyc* est né du projet *CYC*, reprenant une partie de l'information pour la rendre disponible sous licence *open source* et ainsi permettre d'enrichir le web sémantique d'information de sens commun. Plutôt que de tenter de trouver une ou des solutions générales à tous les problèmes d'apprentissage et d'intégration d'information de sens commun, Cyc propose plusieurs modèles correspondants à des contextes bien précis. Puisque l'ambiguïté des termes et des contextes est un problème récurrent dans les domaines de TAL et d'apprentissage automatique, ceci permet d'utiliser le bon type d'information pour la situation désirée (Lenat (1995)).

Une autre base de connaissances de grande envergure est *ConceptNet*, qui se veut un réseau sémantique d'informations de sens commun. Liu et Singh (2004b) ont créé ce projet à partir d'informations générées par les utilisateurs du web. Dans leur cas, les informations sont obtenues par reconnaissance de patrons dans un corpus de phrases, elles-mêmes obtenues par l'entremise de contributeurs du web. La structure de l'ontologie ConceptNet est un ensemble d'associations simples entre deux concepts définis, choisies parmi une liste finie d'associations distinctes. En conséquence de l'origine des informations recueillies, les concepts et associations de ConceptNet sont axés principalement sur leur sémantique textuelle plutôt que sur leur représentation spatiale et fonctionnelle (Liu et Singh (2004a)).

Afin d'être efficace, une ontologie se doit d'être spécifique au domaine d'application pour laquelle elle est utilisée (Russell et Norvig (2010)). Ce faisant, une multitude de domaines d'application différents appellent à un grand nombre d'ontologies spécifiques. L'optique du web sémantique de partager la connaissance est donc compromise par cette sur-spécialisation des ontologies, qui a comme conséquence d'inonder les bases de connaissances partagées d'information spécifique et inutile dans un contexte global (Niles et Pease (2001a)). Le projet

Standard Upper Ontology (*SUO*) parrainé par le IEEE a pour but de trouver une ontologie de haut-niveau définissant un grand nombre de concepts généraux du monde qui serviront ensuite de base pour toutes sortes d'ontologies spécifiques aux domaines et aux applications. Dans le but de combler ces besoins, Niles et Pease (2001b) ont présenté *Suggested Upper Merged Ontology* (*SUMO*), une ontologie de haut-niveau permettant la mise en contexte et la classification de plusieurs ontologies spécifiques. Comme mentionné précédemment, une ontologie de ce type est très utile pour la classification et le partage d'informations sur le web sémantique. Par contre, elle ne peut combler les besoins d'une application de bas-niveau telle que l'introduction d'information sémantique dans les environnements virtuels. Des ontologies répondant aux besoins de ce type d'applications spécifiques sont traitées aux sections 3.2 et 3.3.

3.2 Sémantique des environnements virtuels

Un environnement virtuel est en premier lieu un ensemble d'objets et éléments modélisés en 3D dans le but de représenter une scène virtuelle quelconque. Cette scène peut être statique, dynamique, interactive et même immersive. Plus la scène supporte d'éléments dynamiques interactifs, plus le besoin d'information sémantique est nécessaire afin d'offrir une scène réaliste. Dans cette section, nous nous intéressons aux approches d'ajout d'information sémantique dans les environnements virtuels.

3.2.1 Problématique

La modélisation et l'implémentation de scène virtuelles dynamiques et interactives est un travail coûteux en terme de temps et ressources, vu la complexité possible des scènes. Bien que de nombreux outils et base de données permettent d'accélérer le processus de modélisation graphique, le contenu fonctionnel et interactif de la scène est très rarement adaptable et réutilisable pour d'autres types d'applications (Gutierrez *et al.* (2005)). L'interactivité des éléments de la scène est limitée à ce qui a été prévu et programmé par le concepteur de la scène sur chacun des modèles, permettant ainsi bien peu de créativité pour l'utilisateur et demandant tout de même un travail considérable pour le concepteur.

La solution envisagée à ce problème est l'ajout d'information sémantique dans les environnements virtuels. Celle-ci vise à permettre la réutilisation de l'information fonctionnelle des éléments d'une scène à l'autre. La clé de cette solution est l'utilisation d'information complémentaire, partiellement ou totalement indépendante de la géométrie graphique des objets (Tutenel *et al.* (2008)). Dans le cas de l'animation automatique des scènes, l'information sémantique importante regroupe les fonctionnalités (exemple : fonctions de contrôle, actions

possibles), les conditions d'utilisations (exemple : instruments nécessaires, préconditions) et les conséquences de l'utilisation. Plusieurs approches d'intégration d'information sémantique ont été proposées dans les dernières années, chacune dans son contexte particulier. Nous étudierons les points intéressants de chacune d'elles.

3.2.2 Approches

Le grand nombre de contextes d'application différents engendre une diversité des approches d'intégration d'information sémantique dans les environnements virtuels. Notre but est de parcourir ces approches afin de déterminer lesquelles s'appliquent le plus à notre contexte précis, pour ensuite pouvoir en retirer les idées afin de les adapter à notre problème.

Approche *Smart Objects*

Kallmann et Thalmann (1998) ont été parmi les premiers à proposer l'idée d'ajouter de l'information sémantique lors de la modélisation des objets en 3D. Leur idée principale est d'inclure, sous forme de description de l'objet, toute l'information nécessaire décrivant les interactions possibles avec cet objet. Leur approche, qu'ils nomment *Smart Objects*, consiste à décrire l'information dans un fichier de script dont la seule référence à la géométrie de l'objet est une référence vers le fichier de modèle 3D. Ainsi, ce faible couplage entre le modèle et la description de l'objet permet la réutilisation dans plusieurs applications. Le fichier de script devient alors la représentation de l'objet, dont la géométrie n'est qu'un attribut.

Cette approche propose d'inclure différents niveaux d'information sur l'objet dans le fichier de script, afin d'avoir le plus d'information possible sur celui-ci au même endroit. Certaines propriétés sont intrinsèques, telles que la position relative des parties de l'objet ou les rotations et translations des parties internes lors d'un mouvement, tandis que certaines autres sont relatives aux interactions avec d'autres éléments de la scène. La possibilité la plus intéressante dans notre cas est l'ajout d'information fonctionnelle des objets, permettant ainsi de les lier à des planificateurs avec un certain niveau d'abstraction par rapport au modèle 3D.

Quelques travaux ont repris cette approche intéressante afin de la perfectionner. Peters *et al.* (2003) ont proposé une plate-forme étendue permettant entre autres des interactions plus complexes ainsi qu'impliquant plusieurs agents. De leur côté, Abaci *et al.* (2005) ont plutôt proposé un méthode d'utilisation des Smart Objects pour la planification spatiale dans des environnements virtuels. Leur système est intéressant sur le point de l'ajout d'information sémantique aux objets, offrant un niveau d'abstraction inégalé jusqu'ici. Par contre, malgré la réutilisabilité possible de chacun des objets spécifiques dans différents contextes, leur approche ne permet pas de définir des liens ou des relations entre les types d'objets définis. Il est donc

impossible de structurer la connaissance selon une hiérarchie ou une famille d'objets ayant des fonctionnalités similaires.

Approches ontologiques

Lorsque les notions d'ontologies et de web sémantique commencèrent à devenir plus connues et utilisées, plusieurs personnes ont pensé à s'en servir pour ajouter de l'information sémantique dans les scènes, puisqu'il s'agit d'un format efficace de représentation et de partage d'information.

Otto (2005) propose d'utiliser une structure semblable à une ontologie basée en langage RDF/RDFS afin de représenter l'architecture de l'environnement virtuel avec une couche d'abstraction supplémentaire. À l'aide de ce formalisme, un graphe, nommé *World Model*, est construit à partir des objets présents dans la scène. Chaque objet est représenté par une ressource et peut être composé d'autres ressources représentant ses composantes géométriques. Cette approche est digne de mention car elle est une des premières à utiliser une structure hiérarchique semblable à une ontologie pour l'ajout d'information sémantique dans un environnement virtuel. Par contre, son utilité est plutôt restreinte et l'abstraction est plutôt superficielle pour notre problème, puisque l'information sémantique est directement liée à un objet précis d'une scène spécifique et qu'aucune information générale de sens commun ni de fonctionnalité n'y est présente. Il s'agit plutôt d'une intégration du formalisme ontologique à une scène virtuelle, sans pour autant parler d'information sémantique de sens commun.

Irawati *et al.* (2006) ont été dans les premiers à proposer une véritable ontologie pour l'ajout d'information sémantique dans des environnements virtuels. Leur ontologie, nommée *Spatial Ontology*, consiste en une base de connaissances en deux niveaux, général et spécifique, respectivement indépendant et dépendant du domaine d'application. La base de connaissances est axée sur les objets virtuels ainsi que sur leur disposition géométrique dans une scène 3D. Tous les objets doivent être définis à partir de leur disposition spatiale, par exemple en spécifiant une position et une orientation, et à partir des caractéristiques de dispositions possibles dans la scène. Dans une application pour une scène 3D interactive par exemple, une interrogation de la base de connaissances serait faite à chaque interaction avec l'environnement afin de déterminer si l'action effectuée par l'utilisateur ou un agent intelligent est conforme avec les dispositions géométriques définies par la base de connaissances. Les éléments de la scène doivent être associés à certains types prédéfinis par l'ontologie, ce qui leur confère ces caractéristiques géométriques et spatiales.

Vanacken *et al.* (2007) ont repris ce concept d'ontologie spatiale à deux niveaux afin de diminuer encore plus le couplage entre le domaine d'application et l'ontologie générale, stipulant que cette dernière était trop axée vers le domaine d'application dans les travaux précédents.

Leur approche est intéressante car elle offre une grande flexibilité pour la réutilisabilité dans plusieurs applications. Elle permet également de définir un monde virtuel complètement abstrait de la scène précise dans lequel les concepts peuvent être utilisés. Par contre, l'aspect de spécification de fonctionnalités reliées à la nature des objets est absent, ce qui ne permet pas l'interaction avec des agents externes dans le cadre de la génération automatique d'un plan de scène animée.

Kessing *et al.* (2009) proposent l'approche la plus intéressante à nos yeux, qui consiste en la création d'une ontologie de classes génériques offrant de l'information générale pour les objets dans des scènes virtuelles. Leur idée est similaire à l'approche *Smart Objects* (voir section 3.2.2), mais la structuration de leur information est organisée en hiérarchie de classes et leur système est axé vers la fonctionnalité sémantique des objets plutôt que géométrique. Bien que leur projet soit axé vers l'interaction des personnages dans les jeux vidéo, l'approche est également applicable à tout type d'agent interagissant dans la scène, donc valide pour l'animation automatique également. Ils introduisent la notion de *services*, qui reprend en quelque sorte la notion de fonctionnalités expliquée précédemment, où chaque objet permet, en effectuant une certaine action sur celui-ci, d'obtenir un résultat ou un effet voulu. Pour reprendre leur exemple, un service offert par une machine distributrice est la vente de boissons gazeuses, ayant comme précondition d'être en marche et d'avoir reçu l'argent nécessaire, et comme effet la livraison de la canette de boisson gazeuse. Malgré que leur ontologie ne soit pas disponible au public, nous retenons plusieurs éléments de leur approche, car elle permet vraiment d'inclure l'information sémantique de fonctionnalités des objets, tout en proposant d'utiliser une hiérarchie de classes pour représenter les différents objets.

3.3 Conceptualisation du monde réel

La section 3.2 traite de la façon d'introduire la sémantique dans les environnements virtuels, que ces derniers soient statiques, dynamiques ou interactifs. Par contre, dans le cadre de notre projet, l'aspect sémantique est présent au-delà de la scène virtuelle, dès la phase de représentation conceptuelle abstraite de la scène. Dans l'optique d'être aptes à analyser et transformer de l'information abstraite sur une scène à animer, nous visons à intégrer une forme de connaissance des objets du monde, mais aussi une conceptualisation des actions et des modalités spatiales rattachées à une scène virtuelle. Le but de cette section est de parcourir les différentes approches de conceptualisation de ces aspects du monde réel.

3.3.1 Actions du monde réel

Un concept primordial pour la représentation abstraite d'une scène animée est le concept d'*action*. Dans le vocabulaire de notre projet, les actions représentent le coeur de l'animation, c'est-à-dire ce qui se déroule dans la scène. Il est essentiel de pouvoir représenter les actions abstraites par des concepts spécifiques afin pouvoir les transformer en action concrètes dans un plan d'animation. Quelque approches intéressantes ont déjà été proposées et implémentées, se regroupant en deux philosophies distinctes pour la classification des actions : selon les racines textuelles et selon les racines conceptuelles.

Racines textuelles

Miller (1995) a été un des premiers à proposer une telle classification des mots de la langue anglaise dans sa base de données lexicale nommée *WordNet*. Une certaine conceptualisation des mots est faite par association avec d'autres mots selon des relations sémantiques définies telles que synonymie, antonymie, hyponymie et trois autres. Ce processus permet de définir les mots conceptuellement par rapport aux autres mots de la langue. La partie intéressante pour l'animation consiste en la définition de deux relations spécifiques aux verbes : troponomie et implication. Ces deux relations permettent de catégoriser certains verbes de la langue selon leurs similarités ou leurs implications dans certains contextes. Par contre, la catégorisation des verbes et autres mots de la langue dans *WordNet* est faite uniquement dans un contexte lexical, ce qui s'applique difficilement au contexte de scène virtuelle.

Ma (2006) a utilisé entre autres la base de données WordNet pour concevoir son système *CONFUCIUS*, un logiciel de génération automatique d'animations à partir de texte. Ce dernier utilise la classification des verbes d'actions de *WordNet* afin de traduire une représentation textuelle en animation d'un modèle humain en 3D. Bien que son approche soit intéressante au niveau de l'analyse sémantique des verbes pour les transformer en actions concrètes, celle-ci est principalement axée sur les mouvements intrinsèques des acteurs et ne propose pas de base de connaissances sur l'environnement des acteurs, ni de mécanismes d'interaction avec celui-ci.

Représentation conceptuelle

Une approche différente pour la classification des actions consiste à les catégoriser par signification conceptuelle plutôt que par association directe aux verbes les caractérisant. Plusieurs projets de recherches antérieurs ont exploité et implémenté cette approche.

Bateman *et al.* (1995) ont été parmi les premiers à transformer une ressource lexicale en base de données conceptuelle avec le projet *GUM* (*Generalized Upper Model*), issu du projet

précédent *Penman Upper Model* (Bateman *et al.* (1990)) qui est une base de connaissances pour le traitement automatique de la langue. Le projet *GUM* propose une ontologie permettant de représenter conceptuellement une phrase. Le concept de base défini dans cette ontologie pour représenter une situation est la *configuration*. Cette dernière comporte des attributs tels que *process*, *actor*, *actee* et autres, qui peuvent être essentiels ou non selon le type de configuration. Trois types de configurations sont définis : *Being and Having*, *Saying and Sensing* et *Doing and Happening*. Pour les besoins de notre projet, le type *Doing and Happening* est le plus intéressant car il représente les actions concrètes du monde réel pouvant être exprimées dans un espace physique. Les actions tangibles physiquement, nommées *motions*, sont catégorisées selon deux paramètres principaux : le type de mouvement dans l'espace et l'effet sur un ou plusieurs autres éléments. Les deux types d'effets sont *Affecting* et *NonAffecting*, et pour chacun d'eux, trois types de mouvements sont possibles *SimpleMotion*, *DirectedMotion* et *OrientationChange*. Cette approche de classification est très intéressante car elle permet de catégoriser les types d'actions selon les paramètres qu'elles nécessitent pour être exprimées conceptuellement. Cette classification seule n'est pas suffisante pour représenter des actions spécifiques comme dans le cas d'une animation, mais il s'agit d'une base bien définie.

Baker *et al.* (1998) ont conçu la ressource lexicale *FrameNet*, consistant en une base de données sémantique des mots de la langue anglaise. Par contre, contrairement à *WordNet*, la classification des mots est effectuée sous forme de *frames* sémantiques, auxquels sont rattachés les mots leur correspondant. Ainsi, selon la ou les significations possibles des mots ambigus de la langue, ils peuvent être associés à un ou plusieurs *frames* sémantiques correspondant à leur signification concrète dans le monde réel. Une bonne partie de ce corpus consiste en une classification sémantique des *frames* associés à des verbes, donc représentant des actions du monde réel. Chaque *frame* propose une définition du sens de ce dernier, une liste de *Frame Effects* correspondant à toutes les caractéristiques et attributs possibles associés aux verbes concernés dans une phrase, les relations avec d'autres *frames* ainsi qu'une liste d'unités lexicales consistant en une liste de mots de la langue se rapportant à ce *frame*. Cette classification est intéressante car elle désambiguïse les verbes de la langue pour représenter les actions conceptuellement avec une granularité suffisamment élevée pour une éventuelle représentation concrète.

L'ontologie *ConceptNet* (Liu et Singh (2004a)) présentée précédemment (voir section 3.1.2), propose également une certaine catégorisation des actions sous forme de concepts dans son réseau sémantique de sens commun. Les actions sont représentées par un type de concept nommé *Activity Phrase* et résultent de l'extraction des verbes d'un corpus de phrases. Certaines relations sémantiques sont aussi définies, regroupées sous deux catégories principales,

les événements et les actions, permettant d'indiquer un lien sémantique entre les différentes actions représentées dans l'ontologie. Cette approche est intéressante vu sa granularité très élevée permettant de représenter clairement les actions. Cependant, cette granularité trop élevée ne permet aucune paramétrisation des actions pour une représentation abstraite puisque chaque action concrète possède sa propre représentation. De plus, l'ontologie étant basée uniquement sur l'analyse de contexte dans des phrases, la classification ne tient pas compte du contexte spatial des actions à représenter, un aspect crucial pour la génération d'animation dans une scène virtuelle.

3.3.2 Représentation dans l'espace

Le deuxième aspect important de la conceptualisation du monde réel dans la représentation abstraite de la scène virtuelle est la représentation dans l'espace. En effet, une fois les informations sur les objets et actions de la scène représentés conceptuellement, il est essentiel de pouvoir ajouter de l'information sur leur positionnement spatial, ou dans le cas des actions, leurs modalités spatiales. Quelques projets antérieurs ont étudié cette problématique et proposé des solutions pour la représentation d'information spatiale dans une scène virtuelle.

D'abord, Coyne et Sproat (2001) ont conçu le système *WordsEye* ayant pour but de générer automatiquement une scène virtuelle statique à partir d'une description textuelle. Afin de représenter les informations spatiales des objets de la scène, ce système inclut la notion de relations spatiales (*Spatial Relations*), consistant en la définition d'étiquettes (*spatial tags*) sur les modèle graphiques. Ils associent ensuite la signification des prépositions et autres mots représentant des modalités spatiales à ces *tags* afin que l'idée de la phrase soit bien représentée graphiquement. Cette approche est pratique dans le contexte de leur projet, mais considérant que ce dernier est fortement axé vers la représentation graphique et très peu sur la sémantique, elle n'est pas très intéressante dans notre contexte de représentation sémantique.

La *Spatial Ontology* de Irawati *et al.* (2006) présentée précédemment (voir section 3.2.2) inclut aussi une notion d'information spatiale par la définition du concept de *Spatial Object*, le type de base des objets de la scène virtuelle. Ce concept est décomposé en plusieurs sous-concepts tels que *Spatial Located Object* et *Placeable Object* entre autres, permettant donc de définir des propriétés spatiales à l'objet selon les types auquel il souscrit. Cette approche est utile dans le contexte de placement spatial des objets et d'interaction avec les modèles graphiques 3D, mais ne propose pas d'extension de cette information spatiale sémantique à l'extérieur de la scène graphique, par exemple pour une représentation conceptuelle abstraite.

Finalement, Bateman *et al.* (2010) ont proposé une extension à l'ontologie *GUM* présentée précédemment (voir section 3.3.1) dans le but d'inclure de nouveaux concepts reliés à l'expression d'information spatiale dans une phrase. Cette nouvelle ontologie, nommée *GUM-Space*,

inclut, entre autres, des notions d'emplacement (*GeneralizedLocation*), de route (*GeneralizedRoute*), et plusieurs types de modalités spatiales (*SpatialModality*) incluant des types de contrôle (*Control*) tels que le support (*Support*) et le confinement (*Containment*). Cette ontologie est très intéressante de notre point de vue puisqu'elle définit des concepts relatifs aux modalités spatiales des actions et des objets concernés, mais de façon abstraite puisqu'il s'agit initialement d'une ontologie appliquée à l'analyse de phrases. Elle comporte plusieurs éléments intéressants pour la conception d'un format de représentation conceptuelle abstraite d'une scène virtuelle, car elle combine la catégorisation de base des actions de *GUM* avec l'abstraction des modalités spatiales de l'extension *GUM-Space*.

CHAPITRE 4

PLANIFICATION

Dans le domaine de l'intelligence artificielle, la résolution de problèmes complexes nécessite une approche structurée afin de parvenir à obtenir une solution efficace. Ce chapitre a pour but de présenter la planification comme approche de résolution de problèmes, ainsi que les différentes techniques et algorithmes utilisés afin d'obtenir les solutions souhaitées.

4.1 Définition

La planification est une approche permettant « la détermination d'un plan d'action qui permette d'atteindre un but ». (Russell et Norvig (2010)) Elle utilise les concepts d'*état*, qui consiste en une liste de prédicats atomiques représentant un état courant du monde, appelés littéraux, et d'*action*, qui consiste en une modification des littéraux de l'état courant pour obtenir un nouvel état. Plus de détails sur le formalisme pour la description de ces états et actions seront donnés aux sections 4.2.2 et 4.2.3. L'objectif de la planification consiste à obtenir un *plan*, c'est-à-dire une liste partiellement ou entièrement ordonnée d'actions à effectuer afin d'arriver à un état final désiré, nommé *but*, à partir d'un état initial déterminé.

L'outil permettant la résolution de problèmes de planification est nommé *planificateur*. Il existe plusieurs types de planificateurs implémentant des algorithmes différents, qui sont décrits dans la section 4.3.

4.2 Déclaration du problème

La première étape de la résolution d'un problème de planification est la déclaration de celui-ci. Tout d'abord, le type d'environnement du problème doit être défini, pour ensuite définir les littéraux qui serviront à représenter les états du monde. Puis, l'état initial et le but doivent être formalisés sous forme d'ensembles de littéraux. Finalement, les actions du monde doivent aussi être formalisées selon un formalisme spécifique. Cette section a pour but de présenter les différentes étapes et considérations nécessaires à la déclaration d'un problème de planification.

4.2.1 Environnement

L'environnement d'un problème de planification détermine l'ensemble des conditions dans lequel le planificateur devra résoudre le problème. Dans le cas d'un agent intelligent, cela détermine comment ce dernier peut et doit interagir avec son monde. L'environnement peut être décrit par un groupe de propriétés, dont les principales sont les suivantes (Russell et Norvig (2010)) :

Observabilité Cette propriété, pouvant aussi être nommée **Accessibilité**, correspond à déterminer si l'agent intelligent connaît, à n'importe quel moment dans le temps, toutes les informations relatives à l'état actuel du monde dans lequel il évolue. Un environnement peut être *totalement observable* ou *partiellement observable*. Dans un environnement partiellement observable, l'agent doit souvent emmagasiner de l'information sur ce qu'il observe à différents moments dans le temps, puisqu'il est possible qu'il n'ait plus accès à cette information à un moment ultérieur. Cet effort n'est pas requis lorsque l'agent évolue dans un environnement totalement observable, puisque toute l'information lui est disponible à n'importe quel moment.

Déterminisme Un environnement est considéré comme *déterministe* lorsque l'état du monde suivant peut être déterminé uniquement par l'application des effets d'une action sur l'état actuel. Sinon, il est considéré comme un environnement *stochastique*, ou *non-déterministe*, où certains changements d'un état à son successeur peuvent être engendrés par des événements aléatoires ou non-observables par l'agent. La planification dans un envrionnement stochastique introduit une notion d'incertitude qui complexifie considérablement la résolution du problème (Kaelbling *et al.* (1998)).

Dynamisme Cette propriété correspond à déterminer si l'environnement de l'agent est sujet à changement en tout temps, c'est-à-dire si l'agent doit tenir compte de changements de contexte lors de sa prise de décision. Un environnement *statique* implique qu'aucun changement n'aura lieu à partir du moment où l'agent débute sa prise de décision, c'est-à-dire l'instant où il prend connaissance de l'état actuel, jusqu'au moment où il débute l'exécution de l'action choisie. En contrepartie, un environnement *dynamique* évolue selon un autre facteur que les décisions de l'agent, dans la plupart des cas le temps, et possiblement d'autres agents. Il est aussi possible d'évoluer dans un environnement *semi-dynamique*, qui constitue un environnement où l'état actuel du monde n'est pas modifié dynamiquement, mais que le temps écoulé est tout de même considéré et a une influence sur les décisions de l'agent.

Continuité Cette propriété s'applique à la façon dont les valeurs quantitatives sont représentées dans l'environnement, par exemple les distances ou le temps. Dans un envi-

ronnement *discret*, les valeurs quantifiables sont représentées par un ensemble fini de valeurs possibles. L'environnement possède donc un nombre fini d'états possibles. Il en est le contraire pour un environnement *continu*, où toutes les valeur quantifiables sont gérées comme des nombres réels, et où le nombre d'états possibles de l'environnement est infini.

Séquentialité Un environnement peut être *épisodique* ou *séquentiel*. Dans un environnement épisodique, chacune des décisions de l'agent est basée uniquement sur le contexte actuel puisqu'elle est considérée comme un épisode atomique et n'a aucun effet sur l'état courant. Dans un environnement séquentiel, ou *non-épisodique*, chacune des décisions a des répercussions sur les décisions suivantes puisque l'état courant est modifié par celle-ci. Il est donc nécessaire d'envisager un plan à l'avance plutôt que de ne réfléchir uniquement à la décision actuelle.

Selon les cinq propriétés expliquées ci-haut, les problèmes de planification auxquels nous nous attardons possèdent un envrionnement totalement observable, déterministe, semi-dynamique, discret et séquentiel. Nous pouvons donc traduire ceux-ci en problèmes de planification classique, puisque l'observabilité, le déterminisme et la continuité de l'environnement le permettent (Russell et Norvig (2010)). L'aspect semi-dynamique tient compte du fait que le temps s'écoule dans l'environnement ou l'agent évolue et l'aspect séquentiel prouve le besoin d'utiliser la planification pour résoudre le problème, sans quoi un agent décisionnel simple pourrait faire de même sans avoir à élaborer un plan de résolution.

4.2.2 Formalisme des littéraux et états

Le domaine d'un problème de planification classique est représenté par l'ensemble des littéraux définis. Un littéral est un énoncé atomique représentant une assertion dans le monde défini. Tout état possible du monde est une combinaison de ces littéraux. L'ensemble des combinaisons possibles de littéraux est déterminé par le langage utilisé pour la représentation du problème, puisque celui-ci détermine le formalisme de déclaration et les opérateurs permis.

Fikes et Nilsson (1971) ont conçu le Stanford Research Institute Problem Solver (STRIPS), un planificateur utilisant une syntaxe simple pour définir les littéraux et états possibles du monde. Au fil des années, les algorithmes des planificateurs se sont diversifiés et améliorés, mais le nom STRIPS est toujours couramment utilisé pour décrire le langage formel de spécification des problèmes de planification proposé par Fikes et Nilsson (1971).

Le langage STRIPS propose une syntaxe de déclaration d'un problème de planification qui se divise en trois sections. Les deux premières sections sont la déclaration de l'état initial et la déclaration du but (état final). La troisième section, habituellement plus lourde en contenu,

contient la déclaration des actions. Cette dernière sera expliquée plus en détail à la section 4.2.3.

La représentation d'un état en STRIPS est une conjonction de littéraux positifs. Cela vaut pour tout état à partir de l'état initial jusqu'à l'atteinte du but. Certaines conditions s'appliquent aux littéraux contenus dans ces états afin que la représentation soit valide (Russell et Norvig (2010)) :

1. Les littéraux doivent être atomiques
2. Les littéraux ne doivent pas contenir de variables
3. Les littéraux ne doivent pas contenir de symboles de fonctions
4. Les littéraux doivent être positifs

Le langage STRIPS fonctionne selon l'*hypothèse du monde clos*, selon laquelle tous les littéraux non-mentionnés dans un état sont considérés faux. Un but est considéré comme atteint lorsque tous les littéraux qui y sont déclarés se retrouvent dans l'état actuel, sans tenir compte d'autres littéraux non spécifiés dans ce but qui pourraient aussi être vrais dans l'état actuel.

Les caractéristiques du langage STRIPS en font un langage permettant d'exprimer le but d'un problème de planification de manière simple et de pouvoir le décomposer en une liste de littéraux simples, atomiques et positifs. D'autres langages ont été développés, plusieurs étant des descendants même de STRIPS ou des adaptations, afin de permettre une plus grande flexibilité dans l'expression des littéraux et états. Mcdermott *et al.* (1998) ont présenté le Planning Domain Definition Language (PDDL), un descendant du langage STRIPS qui est toutefois légèrement moins contraint que ce dernier, notamment puisqu'il permet que les littéraux soient négatifs s'ils ont été déclarés ainsi dans la définition initiale du problème (voir section 4.2.3).

Pednault (1989) a proposé le Action Definition Language (ADL), qui, quoique considéré comme une amélioration de STRIPS, possède de nombreuses différences majeures avec ce dernier. Le langage ADL est beaucoup moins contraignant que STRIPS au niveau de la représentation des états et des littéraux, ce qui introduit plusieurs nouvelles possibilités. D'abord, dans la même foulée que le langage PDDL, ADL permet l'utilisation de littéraux négatifs. De plus, il permet d'introduire des disjonctions et des variable quantifiées dans les buts ainsi que des effets conditionnels (voir section 4.2.3), ce qui permet d'exprimer des situations beaucoup plus complexes qu'avec une simple conjonction de littéraux positifs. Par contre, la résolution du problème (voir section 4.3) se retrouve elle aussi complexifiée par les nouvelles possibilités. Une autre différence majeure entre ADL et les autres langages que nous avons présentés est que celui-ci fonctionne selon l'*hypothèse du monde ouvert*, selon

laquelle tous les littéraux non-mentionnés dans un état sont considérés inconnus. Toutes ces caractéristiques du langage ADL en font un langage plus permissif que ses prédécesseurs et bien adapté à la planification dans un monde réaliste (Pednault (1994)).

4.2.3 Formalisme des actions

En planification classique, les actions correspondent aux transitions permettant de passer de l'état actuel vers un nouvel état. Chaque action est définie par un *schéma d'action*. Ce schéma comprend trois parties, soient la *signature*, les *préconditions* et les *effets* (Russell et Norvig (2010)).

La signature est un concept similaire à une signature de fonction dans un langage de programmation standard tel que Java ou C++. Elle déclare le nom de l'action ainsi que les variables qui sont utilisées dans sa définition. Certains langages, tels que PDDL et ADL, supportent les types pour les variables. Ceux-ci sont énoncés dans la signature de l'action en même temps que la variable. D'autres langages, tels que STRIPS, n'associent pas de type aux variables. Le cas échéant, les types sont habituellement présentés sous forme de littéraux et se retrouveront dans la liste de préconditions.

La liste de préconditions est une combinaison de littéraux servant à déterminer l'*applicabilité* d'une action. En effet, afin que l'action puisse être appliquée, la liste de préconditions doit être entièrement respectée, c'est-à-dire présente dans l'état courant. Lorsque les préconditions sont sous forme de conjonction de littéraux positifs, tel que c'est le cas dans le langage STRIPS, cette vérification se résume à confirmer que chacun de ces littéraux se retrouvent dans l'état courant. Par contre, lorsque la liste de préconditions peut contenir des disjonctions ou des littéraux négatifs, la vérification est plus complexe.

La liste des effets est aussi une combinaison de littéraux. Ceux-ci représentent les modifications à apporter à l'état courant à la suite de l'application de l'action. La liste contient deux sections : la *liste de suppression* et la *liste d'ajouts*. La liste de suppression correspond à l'ensemble des littéraux qui doivent être retirés de l'état courant suite à l'application de l'action. Dans un langage tel que STRIPS, la liste de suppression représente plutôt une liste de négation, en indiquant les littéraux qui cessent d'être vrais. Puisqu'il fonctionne selon l'hypothèse du monde fermé, chaque littéral de la liste de suppression non présent dans l'état courant n'a aucun effet. Par contre, dans un langage fonctionnant selon l'hypothèse du monde ouvert tel que ADL, les littéraux négatifs peuvent se retrouver dans l'expression des états. Chaque littéral de la liste de suppression doit alors être appliqué à l'état courant. Ensuite, la liste d'ajouts correspond à l'ensemble des littéraux qui doivent être ajoutés à l'état courant suite à l'application de l'action. Dans le langage STRIPS, ces littéraux peuvent être ajoutés tels quels tandis que dans le langage ADL, ils doivent être combinés avec ceux de l'état

courant afin de s'assurer que ceux qui annulent l'effet d'un autre littéral soient combinés et supprimés.

```
(:action place-in
  :parameters (
    ?human - Human
    ?object - Object
    ?container - Container
    )
  :precondition (and
    (at ?human ?container)
    (holding ?human ?object)
    )
  :effect (and
    (containing ?container ?object)
    (not (holding ?human ?object))
    )
  )
```

Figure 4.1 Exemple de définition d'une action en PDDL

L'exemple de la figure 4.1 illustre la définition de l'action *place in* en PDDL, qui signifie de placer un objet dans un contenant. La signature comporte le nom de l'action ainsi que ses paramètres, dans ce cas-ci trois objets de types *Human*, *Object* et *Container*. La liste de préconditions contient deux littéraux : un représentant que l'humain doit être au même endroit que le contenant et le second signifiant que l'humain doit tenir l'objet. Une fois l'action effectuée, deux effets sont appliqués sur l'état courant : le contenant contient maintenant l'objet et l'humain ne tient plus l'objet.

4.3 Résolution du problème

Une fois le problème de planification déclaré à l'aide d'un formalisme précis, l'étape suivante est la résolution du problème. Il existe de nombreux algorithmes pour la résolution de problèmes de planification (Russell et Norvig (2010)). Nous présentons ici quelques-uns de ces algorithmes qui sont applicables dans un contexte de planification classique (voir section 4.2.1) :

- Exploration dans un espace d'états
- Graphes de planification
- Problèmes de satisfaction de contraintes
- Planification à ordre partiel
- Réseaux hiérarchisés de tâches

4.3.1 Exploration dans un espace d'états

L'exploration dans un espace d'états est un algorithme qui consiste à explorer l'ensemble des états atteignables par l'application des actions sur l'état courant. Le problème de planification est donc réduit à un problème d'exploration. Il existe deux approches principales à l'exploration dans un espace d'états : l'exploration *progressive* et l'exploration *régressive*.

L'exploration progressive dans un espace d'états consiste à commencer la recherche à partir de l'état initial et à parcourir les états atteignables par exécution des actions applicables jusqu'à l'atteinte du but. À la base, ce type d'exploration est considéré comme inefficace dû au nombre habituellement très élevé de possibilités, représentés par des noeuds dans le graphe d'exploration. De plus, puisque la recherche est progressive à partir de l'état initial, elle n'est pas axée vers les buts. La recherche demande donc un nombre très grand de ressources et de temps d'exécution dans des situations sans grande complexité.

L'exploration régressive est orientée inversement à l'exploration progressive, c'est-à-dire qu'elle consiste à commencer la recherche à partir de l'état final, ou but, et à parcourir les états ayant permis d'atteindre cet état par l'exécution inverse des actions. Plutôt que de tester l'applicabilité des actions, il faut plutôt vérifier que les effets des actions sont cohérents avec l'état courant afin de pouvoir revenir à l'état précédant en appliquant inversement les préconditions et en supprimant les effets.

Qu'il s'agisse d'exploration progressive ou régressive, aucune des deux approches ne peut être assez efficace sans l'utilisation d'une heuristique (Russell et Norvig (2010)). La résolution d'un problème de planification par exploration dans un espace d'états considère donc toujours l'utilisation d'une heuristique, ce qui incite la grande majorité des systèmes actuels à utiliser une approche d'exploration progressive, puisque son algorithme simplifie de beaucoup la construction d'une bonne heuristique. Plusieurs planificateurs actuels utilisent cette approche dont HSP (Bonet (1983), Bonet et Geffner (2001)), Prodigy (Carbonell *et al.* (1991)), FF (Hoffmann et Nebel (2001)) et plus récemment Fast-Downward (Helmert (2006)) et HPLAN-P (Baier *et al.* (2009)), se différenciant tous par l'heuristique utilisée pour leur algorithme d'exploration.

4.3.2 Graphes de planification

Les graphes de planification sont des structures de données de type graphe orienté qui permettent à un algorithme d'exploration d'obtenir de meilleures estimations heuristiques. Puisque l'arbre de possibilités à partir de l'état initial croît de manière exponentielle en rapport au nombre d'actions du plan, il n'est pas du tout efficace de tenter de le construire entièrement. L'approche d'un graphe de planification en propose donc une approximation

de taille polynomiale et facilement constructible. Elle permet d'estimer sans surestimation le coût pour atteindre le but à partir de l'état initial et aussi de déterminer si le but est atteignable. Elle ne garantit pas de déterminer dans tous les cas s'il est atteignable, mais la prédiction est sans erreur lorsqu'elle est effectuée.

Blum et Furst (1997) ont présenté un algorithme du nom de GRAPHPLAN permettant l'extraction d'un plan directement à partir d'un graphe de planification. Son fonctionnement consiste à obtenir un niveau où tous les littéraux des buts sont représentés sans lien d'exclusion mutuelle. Ensuite, un algorithme avec base heuristique effectue une exploration régressive (voir section 4.3.1) afin d'obtenir le plan. Si le plan est impossible à obtenir, des niveaux sont rajoutés jusqu'à l'obtention de la solution. Dans le cas où celui-ci est toujours introuvable et que l'heuristique ne peut plus être respectée, l'extraction du plan à partir de GRAPHPLAN est considérée impossible.

4.3.3 Planification à ordre partiel

La planification à ordre partiel (POP) est une variation de l'approche d'exploration dans un espace d'états. Dans cette approche, le problème est toujours considéré comme un problème d'exploration, mais l'espace d'états est transformé en un espace de plans. Ceci est rendu possible par l'identification des actions ou sous-plans indépendants, c'est-à-dire des actions n'interférant pas entre elles par leurs préconditions ou effets, donc pouvant être effectuées dans un ordre quelconque ou parallèlement. Le plan final est dit partiellement ordonné, plutôt que totalement ordonné dans le cas de l'exploration dans un espace d'états, et est représenté par un ensemble d'actions et de contraintes. Ces dernières spécifient explicitement que deux actions sont dépendantes l'une de l'autre, et dans l'ordre qu'elle doivent apparaître dans le plan final. Le reste de l'ordonnancement du plan est flexible et possiblement parallélisable. Penberthy et Weld (1992) ont d'ailleurs présenté UCPOP, un planificateur exploitant les avantages de la planification à ordre partiel.

4.3.4 Problèmes de satisfaction de contraintes

Une avenue possible de la résolution d'un problème de planification est la traduction de ce dernier en un *problème de satisfaction de contraintes* (Constraint Satisfaction Problem, ou CSP). Un CSP consiste en la définition d'un problème par un ensemble de variables soumises à un domaine de valeurs possibles et à un ensemble de contraintes sur ces variables. Cette technique de résolution est efficace lorsque le problème de planification est borné, c'est-à-dire qu'il s'agisse de trouver un plan de longueur déterminée. Kautz *et al.* (1996) ont présenté une méthode de *satisfiabilité booléenne* (SAT) connue sous le nom de SATPLAN pour la résolution

de problème de planification. De plus, Do et Kambhampati (2001) ont proposé le système GP-CSP, permettant de traduire un graphe de planification en problème de satisfaction de contraintes.

4.3.5 Réseaux hiérarchisés de tâches

Une approche différente à la résolution de problèmes de planification est la décomposition du problème de manière hiérarchique. La planification hiérarchique diffère des autres approches présentées jusqu'ici par le fait qu'elle ne considère pas les actions comme un ensemble fini d'éléments atomiques, mais plutôt en décrivant des *actions de haut niveau* (High-Level Actions, ou HLA). Celles-ci correspondent à un ensemble d'*affinements*, consistant en des actions simples, dites *actions primitives*, ou à d'autres actions de haut niveau. Ceci permet d'établir une hiérarchie dans les actions et ainsi de décomposer le plan en une arborescence d'actions des deux types. Cette approche de planification est appelée planification par *réseaux hiérarchisés de tâches* (Hierarchical Task Networks, ou HTN) et permet l'introduction de concepts d'implémentations d'actions différentes des approches exploratives simples comme la récursivité, par exemple (Russell et Norvig (2010)). Erol *et al.* (1995) ont démontré par leur compréhension sémantique des algorithmes HTN que le principal avantage de ces derniers est d'être beaucoup plus expressif qu'un algorithme standard d'exploration dans un espace d'états.

4.4 Synthèse critique

Pour les besoins de notre projet, nous devons trouver un planificateur permettant d'exprimer le problème le plus simplement possible, tout en implémentant un algorithme suffisamment efficace pour trouver une solution optimale à la majorité des problèmes. En effet, pour que les animations soient crédibles et faciles à interpréter en bout de ligne, elles doivent s'en tenir à la plus simple expression possible de la description initiale.

L'aspect le plus important de l'utilisation de la planification dans notre projet est la définition du problème, puisque le but global est de transformer une description conceptuelle abstraite en problème de planification pour ensuite le résoudre. Considérant qu'une scène virtuelle enrichie sémantiquement combinée à une ontologie de sens commun peut comporter un très grand nombre d'informations, un formalisme complexe comme ADL fonctionnant selon l'hypothèse du monde ouvert est déconseillé, puisqu'il signifie un travail important pour la formulation du problème tout en complexifiant sa résolution. Un formalisme de type STRIPS est suffisant pour représenter le problème tout en simplifiant la résolution de ce dernier. Le langage le plus fréquemment utilisé dans ces conditions est PDDL.

La résolution du problème de planification est bien sûr primordiale à l'atteinte des buts du projet. Bien que les scènes virtuelles enrichies sémantiquement puissent contenir un grand nombre d'informations, nous n'anticipons pas de plans possédant un très grand nombre d'actions puisque les animations à générer correspondent à des descriptions simples et concises. Le défi se retrouve davantage du côté du nombre élevé d'actions possiblement définies dans le domaine du problème plutôt que de la longueur des plans à générer. L'utilisation de planification à ordre partiel et de réseaux hiérarchisés de tâches ne semble donc pas appropriée pour notre type de problème. Un algorithme comme GRAPHPLAN, par contre, se trouverait assez utile pour nos besoins. Le planificateur Blackbox (Kautz et Selman (1998)) propose de traduire un problème en formalisme STRIPS en problème de satisfaction de contraintes tout en combinant à un algorithme GRAPHPLAN pour sa résolution. Celui-ci est disponible en ligne et accepte les problèmes formulés selon le formalisme STRIPS, en langage PDDL et acceptant le typage des objets. Il répond donc à nos besoins par rapport au choix de planificateur.

CHAPITRE 5

SOLUTION PROPOSÉE

Afin de résoudre la problématique de génération automatique d'un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite (voir chapitre 2), nous proposons une solution qui combine un format de représentation abstraite ainsi qu'une base de connaissances de type ontologie pour permettre la formulation d'un problème de planification. Ce chapitre a pour but de présenter cette solution détaillée.

La présentation de notre solution est divisée en quatre sections principales. D'abord, nous exposons le format proposé pour la représentation conceptuelle abstraite de la scène. Nous exposons ensuite un second format pour la représentation de l'information contenue dans la scène statique. Puis, nous présentons la structure de la représentation de la connaissance utilisée pour l'ajout d'information de sens commun. Finalement, nous expliquons le processus de traitement de l'information permettant de formuler le problème de planification.

5.1 Représentation conceptuelle abstraite de la scène animée

La représentation conceptuelle abstraite que nous proposons est un format XML de description de scène inspiré de l'ontologie *GUM-Space* (Bateman *et al.* (2010)). Ce format comporte une structure précisément définie et validée basée sur l'ontologie *GUM-Space* ainsi que des liens vers la ressource *FrameNet* (Baker *et al.* (1998)) et notre propre ontologie. Nous présentons d'abord les besoins spécifiques de cette représentation ainsi que l'abstraction des informations ayant été effectuée, puis les éléments basés sur l'ontologie *GUM-Space* ainsi que la représentation des éléments de connaissance. Finalement, nous exposons le format XML résultant ainsi que son schéma de validation.

5.1.1 Besoins spécifiques et abstraction des informations

Afin de parvenir à la définition d'un format de représentation conceptuelle abstraite de la scène, il est nécessaire d'identifier les informations importantes devant être représentées pour la description d'une scène virtuelle animée. Ces informations se résument aux éléments essentiels définissant la scène et devant être spécifiés pour permettre une compréhension de celle-ci.

Dans un premier temps, le point central de la scène animée est l'action principale de celle-ci. Lorsque le but de la scène virtuelle animée est de décrire une situation simple, l'action

effectuée dans cette situation est le point le plus important à représenter. Il s'agit donc de pouvoir représenter l'action principale par un concept défini, permettant ensuite de spécifier des paramètres pour la réalisation de cette action.

Chacun des éléments physiques présents dans la scène doit pouvoir être identifié et représenté conceptuellement afin de pouvoir en déduire sa nature et ses fonctionnalités. Il est donc nécessaire de fournir un identifiant pour chacun des différents objets, en plus de spécifier sa nature à l'aide d'un concept provenant d'une base de connaissances.

Les éléments ayant les rôles prédominants dans l'action principale doivent être identifiés. L'élément acteur de l'action doit alors pouvoir être spécifié comme tel, ainsi qu'un élément affecté par cette même action, au besoin. Dans les paramètres de l'action doivent donc se retrouver des références à ces deux éléments principaux.

Une fois l'action principale et ses éléments identifiés, de l'information supplémentaire sur celle-ci est nécessaire afin de décrire son déroulement dans la scène virtuelle. Ainsi, de l'information spatiale doit pouvoir être spécifiée et associée à l'action principale. Cette information peut représenter plusieurs aspects, comme le lieu du déroulement de l'action, la direction ou un itinéraire emprunté lors d'un mouvement, entre autres. Ces aspects doivent pouvoir être définis selon les objets concernés de la scène puisque la représentation 3D est abstraite dans le cadre de notre projet. Par exemple, un itinéraire, que nous appelons *route* dans notre contexte, emprunté lors d'un mouvement peut être « du comptoir à la table » et une direction peut se limiter à « vers le mur ». Des exemples de descriptions abstraites de ce type d'information sont présentés à la section 5.1.2. Certaines modalités spatiales par rapport à ces objets doivent aussi pouvoir être représentées pour désambiguïser leur rôle. Par exemple, un objet de type « commode » étant impliqué lors d'une action peut contenir des objets, servir de support à d'autres objets ou simplement être le lieu d'interaction de l'acteur sans autre objet impliqué. Il est donc nécessaire de spécifier précisément le rôle des éléments dans le déroulement spatial de l'action principale de la scène animée.

Finalement, comme quelques ressources sémantiques et bases de connaissances seront utilisées afin de représenter conceptuellement tous les éléments de la scène, un mécanisme doit prévoir la liaison avec celles-ci dans le format de représentation. Puisque la portée de notre projet ne prévoit pas d'enrichir les ressources externes avec de l'information sémantique supplémentaire, un simple mécanisme de références vers les concepts nécessaires est suffisant pour la compréhension du problème et l'analyse de l'information abstraite.

5.1.2 Utilisation de *GUM-Space*

Nous avons présenté l'ontologie *GUM-Space* à la section 3.3.2 en soulignant les aspects principaux de représentation d'information spatiale et de classification de base des actions.

Comme mentionné dans cette section de notre revue de littérature, ces aspects sont très intéressants pour la création d'une représentation conceptuelle abstraite. Après étude de différentes options valables, nous avons choisi d'utiliser les concepts définis par *GUM-Space* pour définir la structure de notre format de représentation.

Motivation

Notre décision repose sur le fait que l'ontologie *GUM-Space*, version améliorée de l'ontologie GUM (Bateman *et al.* (1995), voir section 3.3.1), est celle, parmi tous les travaux effectués dans le domaine, définissant le plus de concepts utiles à la représentation conceptuelle abstraite d'une scène et utilisables dans le but d'illustrer cette scène dans un espace virtuel. Nous avons retenu deux aspects principaux de *GUM-Space* pour notre format de représentation : la catégorisation des actions de type mouvement dans l'espace (*motion*) en six types de base et la définition de concepts spatiaux permettant d'exprimer diverses informations sur la disposition spatiale de la scène ainsi que des actions et éléments la composant.

En premier lieu, la division des actions de type mouvement en six types de base distincts permet de définir les paramètres spécifiques à chacun de ces types. Le fait que la division soit faite selon le type de mouvement dans l'espace ainsi que l'effet sur un ou plusieurs autres éléments de la scène permet de très bien identifier les paramètres nécessaires. Par exemple, une action de type *AffectingDirectedMotion* indique que la description de l'action comporte une direction ou une route et qu'elle implique un autre objet de la scène. Par contre, une action de type *NonAffectingSimpleMotion* ne comporte ni route, ni direction et n'implique pas d'élément de la scène autre que l'acteur même. Nous reprenons cette catégorisation dans notre format de représentation car elle offre un excellent niveau d'abstraction des actions du monde réel, qui peuvent être identifiées uniquement selon leurs paramètres spatiaux et fonctionnels.

Ensuite, l'aspect le plus intéressant de l'ontologie *GUM-Space* par rapport à notre projet réside dans la définition de nombreux concepts spatiaux. Étant initialement destinée à la représentation conceptuelle de phrases, elle définit des concepts dans le but de représenter conceptuellement de l'information spatiale abstraite. Un concept de base de représentation d'un emplacement spatial (*GeneralizedLocation*) est défini et largement utilisé dans la définition sémantique des autres concepts de l'ontologie. Celle-ci prévoit également des concepts de route (*GeneralizedRoute*) et de chemin (*GeneralizedPathLocation*), ce dernier représentant un emplacement à atteindre lors du parcours d'une route, autre que le point de départ et la destination. Ces trois concepts sont très utiles pour la génération d'une animation, puisque cette information doit absolument être transmise de la phase conceptuelle à la phase d'animation. Le concept le plus complexe et complet que nous reprenons de cette ontologie

est celui de modalité spatiale (*SpatialModality*). Ce dernier est divisé en une panoplie de concepts spécifiques permettant d'exprimer toutes sortes d'information spatiale sur les objets de la scène, de relations spatiales entre ces mêmes objets et de spécifications spatiales sur les actions se déroulant dans la scène. Nous n'utiliserons qu'une partie de ces concepts dans le cadre de notre projet, mais cette collection permet d'affirmer que l'ontologie *GUM-Space* est tout à fait adaptée pour la description spatiale d'une scène virtuelle dans une représentation conceptuelle abstraite.

Structure de *GUM-Space*

Le concept de base de l'ontologie *GUM-Space* pour représenter une situation est la *configuration*. Plusieurs types de configurations y sont définis et représentent chacun un type d'action différent dans la situation exprimée. Tel qu'expliqué précédemment (voir section 3.3.1), le type de configuration intéressant pour nous est *Doing and Happening*, particulièrement ses sous-types *AffectingMotion* et *NonAfffectingMotion*.

Pour exprimer une situation à l'aide des concepts définis dans l'ontologie, celle-ci doit être décortiquée selon ses différents éléments. Ces éléments constituent par exemple le type d'action, l'acteur, les éléments affectés ou les modalités spatiales de l'action. Premièrement, le type de configuration est défini et ensuite les éléments y sont associés afin de compléter l'information relative à la situation décrite.

Afin de faciliter la compréhension de la structure des configurations dans *GUM-Space*, nous proposons d'analyser une phrase à titre d'exemple. L'exemple de la figure 5.1 illustre la représentation de la phrase « He is dancing in the street » à l'aide du formalisme de *GUM-Space*.

```
NonAffectingSimpleMotion 'nasm1'
     actor SimpleThing 'he' He
     processInConfiguration Process 'dancing' is dancing
     placement GeneralizedLocation 'gl1'
          relatum SimpleThing 'street' the street
          hasSpatialModality Containment 'con1' in
```

Figure 5.1 Représentation d'une configuration de type NonAffectingSimpleMotion

Le but de cette représentation est de transformer une phrase, donc une représentation textuelle concrète, en un formalisme conceptuel afin de permettre de tirer profit des différents éléments de la situation exprimée par la phrase. Le premier élément identifié dans la représentation est le type de configuration. Celle-ci est de type *NonAffectinSimpleMotion*,

ce qui signifie que l'action décrite dans cette situation n'affecte pas d'objets de la scène autres que l'acteur et ne nécessite pas d'information relative à une direction ou une route. Les deux seuls éléments devant absolument être identifiés sont donc l'acteur et l'action elle-même. D'abord, le premier sous-élément de la configuration identifié est l'acteur. Le type de ce dernier est spécifié et un identifiant unique lui est attribué. Dans l'exemple présenté, 'He' correspond à l'acteur et son type est *SimpleThing*. Ensuite, le deuxième sous-élément identifié correspond à l'action concrète (*process*). Dans cette phrase, l'action correspond au verbe 'dancing'. À cet endroit, toutes les information obligatoires à la description d'une configuration de type *NonAffectingSimpleMotion* ont été identifiées et représentent la locution « He is dancing ». Par contre, dans ce cas-ci, de l'information supplémentaire doit être fournie afin de représenter l'information contenue dans la locution « in the street » qui complémente la première. Un troisième sous-élément est donc ajouté à la description de la configuration. Celui-ci correspond à un emplacement (*placement*) de type *GeneralizedLocation*, qui signifie un emplacement spatial. Il est caractérisé par un *relatum* et une modalité spatiale (*SpatialModality*) qui dépendent l'un de l'autre. Le *relatum* indique l'objet de la scène en relation spatiale avec l'action décrite et la modalité spatiale indique de quelle façon ils sont en relation. Dans le cas présent, le relatum correspond à 'street' et la modalité spatiale est de type *Containment*, indiquant que l'action se déroule 'dans' la rue.

Il est à noter que le formalisme de *GUM-Space* identifie tous les objets et acteurs d'une situation en tant qu'objets de type *SimpleThing* et les actions en tant qu'objets de type *Process*. L'identifiant attribué dans les exemples représente l'objet concerné dans la phrase pour faciliter la compréhension, mais n'introduit aucune information sur la nature réelle de l'objet. Il en est de même pour la représentation des actions, qui ne sont pas associées à une représentation sémantique précise. Cet aspect de notre format de représentation est abordé à la section 5.1.3.

Afin d'illustrer une possible variation du formalisme de *GUM-Space* par rapport au type de configuration à représenter, nous proposons un second exemple. La figure 5.2 illustre la représentation de la phrase « He puts the ball in the box ».

Nous reconnaissons ici le même formalisme qu'à l'exemple de la figure 5.1, à l'exception près de quelques éléments. D'abord, le type de configuration n'est pas le même puisqu'il ne s'agit pas du même type d'action. Cette configuration est de type *AffectingDirectedMotion*, ce qui signifie que l'action décrite dans cette situation affecte un objet de la scène autre que l'acteur et spécifie de l'information relative à une direction ou une route. Ensuite, les éléments obligatoires *actor* et *process* sont toujours présents, mais nous pouvons constater l'ajout d'un élément *actee*, qui signifie l'élément affecté par cette action. Dans ce cas-ci, il s'agit de l'objet 'ball'. Puis, l'information relative à la route exprimée dans la situation est décrite sous la

```
AffectingDirectedMotion 'adm1'
     actor SimpleThing 'he' He
     processInConfiguration Process 'putting' puts
     actee SimpleThing 'ball' the ball
     route GeneralizedRoute 'gr1'
          destination GeneralizedLocation 'gl1'
               relatum SimpleThing 'box' the box
               hasSpatialModality Containment 'con1' in
```

Figure 5.2 Représentation d'une configuration de type AffectingDirectedMotion

forme d'un élément de type *GeneralizedRoute*. Ce type d'attributs peut contenir plusieurs emplacements spatiaux tels une source (*source*), une destination (*destination*) ou un chemin (*pathPlacement*). Dans le cas présent, puisqu'aucun emplacement initial ni chemin particulier de la balle n'est spécifié, l'attribut route possède uniquement une destination, représentant la locution « in the box ». Cette destination est du même type *GeneralizedLocation* que nous avons présenté précédemment, donc possède 'box' comme *relatum* et *Containment* comme modalité spatiale.

À la suite de la présentation de ces deux exemples, nous pouvons constater que le formalisme défini par l'ontologie *GUM-Space* est très bien adapté à plusieurs besoins du format de représentation identifiés à la section 5.1.1. Nous avons donc retenu celui-ci et l'avons adapté afin de servir de structure pour notre propre format de représentation.

5.1.3 Format de représentation

À la suite de l'analyse des besoins spécifiques et du choix de l'ontologie *GUM-Space* comme structure de base, nous avons défini un format de représentation conceptuelle abstraite de la scène virtuelle en langage XML. Nous avons choisi ce format car nous avions besoin d'un format pouvant représenter des informations sous forme hiérarchique, faire des liens vers des ressources externes et être validé rigoureusement selon un formalisme que nous pouvons définir, tout en n'étant pas restreint à l'utilisation du formalisme exact de l'ontologie *GUM-Space*. Notre format inclut ainsi des mots-clés basés sur la structure des configurations de *GUM-Space* ainsi que des liens vers les bases de connaissances *FrameNet*, *GUM-Space* ainsi que notre propre ontologie. Ces liens servent à spécifier la nature des éléments à représenter dans la scène animée.

Représentation des éléments de connaissance

Afin de permettre la représentation de concepts précis dans notre format de représentation, plusieurs bases de connaissances ont été utilisées. Nous avons déjà présenté l'ontologie *GUM-Space* qui allait servir de base pour la structure de notre format, nous présentons maintenant les bases de connaissances servant pour la représentation des éléments de la scène.

D'abord, pour la représentation de l'action concrète se déroulant dans la scène, nous utilisons la classification de *FrameNet* (Baker *et al.* (1998)), présentée à la section 3.3.1. En utilisant cette classification, nous pouvons identifier l'action se déroulant dans la scène par un *frame* sémantique précis ne tenant pas compte du verbe d'action mais bien de son sens pratique. *FrameNet* définit le concept *Event*, représentant le *frame* de base pour la représentation des actions. Dans notre format de représentation, un *frame* dérivant du concept *Event* par scène décrite est identifié et associé à l'élément *process* (voir section 5.1.2). Comme le nom des frames et leur signification concrète est bien défini par *FrameNet*, nous pouvons utiliser uniquement les noms comme identificateurs.

Ensuite, une seconde base de connaissances référencée dans notre format est l'ontologie *GUM-Space* elle-même. Bien que la plupart des concepts de cette dernière sont utilisés comme mots-clés XML, certains concepts doivent quand même être référencés, tel le type des modalités spatiales. Plus de détails sur l'utilisation de ces concepts sont donnés à la sous-section suivante.

Enfin, l'information de notre propre ontologie des objets du monde réel, présentée à la section 5.3 est également représentée. Pour chaque objet présent dans la scène, une référence vers une classe de notre ontologie est spécifiée afin de déclarer le type concret de l'objet en question.

Adaptation de *GUM-Space*

Le formalisme que nous proposons pour la représentation conceptuelle abstraite consiste en une adaptation de la structure des configurations de *GUM-Space*, dont des exemples ont été donnés à la section 5.1.2. Afin d'illustrer de quelle façon ce format a été adapté, nous reprenons un exemple similaire à celui de la figure 5.1 afin de la traduire en une représentation en langage XML. Ceci est présenté à la figure 5.3, qui illustre la représentation de la phrase « He is dancing on the ground ».

D'abord, le concept de configuration y est explicitement défini, avec un attribut spécifiant le type de celle-ci. Ensuite, nous y retrouvons plusieurs mots-clés du formalisme de *GUM-Space* ayant été convertis en éléments XML tels que *process*, *actor*, *placement*, *relatum* et *spatialModality*. De plus, l'élément complexe *placement* dans notre exemple spécifie son

```
<sceneDeclaration>
  <configuration xsi:type="NonAffectingSimpleMotion">
    <process>fn:Self_motion</process>
    <actor>
      <name>He</name>
      <class>ph:Human</class>
    </actor>
    <placement xsi:type="GeneralizedLocation">
      <relatum>
        <name>ground</name>
        <class>ph:Ground</class>
      </relatum>
      <spatialModality>
        <type>gum:Support</type>
      </spatialModality>
    </placement>
  </configuration>
</sceneDeclaration>
```

Figure 5.3 Représentation XML d'une configuration de type NonAffectingSimpleMotion

type comme étant *GeneralizedLocation*. Les éléments de connaissance présentés dans la sous-section précédente sont référencés par les préfixes *fn*, *ph* et *gum*, représentant respectivement *FrameNet*, notre propre ontologie et *GUM-Space*. L'action décrite dans la scène est représentée par le type de *process* et correspond au *frame Self-motion* de *FrameNet*. Les objets physiques de la scène, dans ce cas-ci l'acteur 'He' et le sol, sont représentés par un identifiant (*name*) et une classe de notre ontologie (*class*). Finalement, la modalité spatiale de support est exprimée par un lien vers le concept *Support* de l'ontologie *GUM-Space*.

De façon similaire à ce dernier exemple, nous pouvons traduire l'exemple de la figure 5.2 en une représentation XML. Ceci est présenté à la figure 5.4, qui illustre à nouveau la représentation de la phrase « He puts the ball in the box ».

Nous y retrouvons à nouveau les concepts de *GUM-Space* convertis en éléments XML tels que *actee*, *route* et *destination*. De plus, les éléments complexes *route* et *destination* dans notre exemple spécifient leur type comme étant respectivement *GeneralizedRoute* et *GeneralizedLocation*. L'action décrite correspond au *frame Placing*, qui signifie le placement d'un objet à un endroit précis. Les objets physiques de la scène, dans ce cas-ci l'acteur 'He', la balle et la boîte, sont toujours représentés par un identifiant et une classe de notre ontologie. Finalement, l'expression de la modalité spatiale de confinement est exprimé par un lien vers le concept *Containment* de l'ontologie *GUM-Space*.

```
<sceneDeclaration>
  <configuration xsi:type="AffectingDirectedMotion">
    <process>fn:Placing</process>
    <actor>
      <name>He</name>
      <class>ph:Human</class>
    </actor>
    <actee>
      <name>ball</name>
      <class>ph:Ball</class>
    </actee>
    <route xsi:type="GeneralizedRoute">
      <destination xsi:type="GeneralizedLocation">
        <relatum>
          <name>box</name>
          <class>ph:Box</class>
        </relatum>
        <spatialModality>
          <type>gum:Containment</type>
        </spatialModality>
      </destination>
    </route>
  </configuration>
</sceneDeclaration>
```

Figure 5.4 Représentation XML d'une configuration de type AffectingDirectedMotion

Schéma de validation

Le format XML proposé encapsule l'information abstraite de description de la scène dans un formalisme XML facile à traiter, autant du point de vue de compréhension par l'humain que de traitement par l'ordinateur. Dans cette optique, le format doit être bien balisé afin de permettre une compréhension optimale de tous les éléments de la scène représentée. Nous avons donc créé un schéma de validation XML en format XSD (*XML Schema Definition*) permettant de définir précisément la forme et le contenu du fichier XML de description de scène. Ce schéma se retrouve à l'Annexe A.

La hiérarchie de notre schéma de validation respecte celle des classes de *GUM-Space*. En effet, le type *configuration* est le type de base de toutes les configurations définies dans le schéma. Il définit uniquement un élément de type *process*. Ses sous-types définissent ensuite les éléments qui sont nécessaires à leur description. Par exemple, les configurations de type *AffectingAction* définissent obligatoirement deux éléments *actor* et *actee* de type *SimpleThing*. Puis, un sous-type de *AffectingAction*, par exemple *AffectingDirectedMotion*, peut définir d'autres éléments tels que *placement*, *motionDirection*, *route* et *direction*, certains

étant spécifiques à ce type de configuration.

En plus des types de configurations, les balises sont données au sujet des types spatiaux tels que *GeneralizedRoute*, *GeneralizedPathLocation* et *GeneralizedLocation*, ainsi que des modalités spatiales (*SpatialModality*). Par exemple, le type *GeneralizedLocation* définit obligatoirement un *relatum* de type *SimpleThing* et une *spatialModality*. De son côté, le type *GeneralizedRoute* ne définit aucun élément obligatoire, mais ne peut que définir des éléments *source* et *destination* de type *GeneralizedLocation* ainsi qu'un élément *pathPlacement* de type *GeneralizedPathLocation*.

Le schéma de validation défini permet premièrement de s'assurer de la validité des fichiers XML de description de scène. Il s'agit d'une première étape pour vérifier la validité de la scène décrite dans notre format de représentation. Ensuite, le schéma définit les balises pour la suite du processus de génération du plan d'animation. En effet, l'existence de celui-ci permet de s'assurer une compréhension toujours uniforme lors du processus d'analyse de la scène décrite et ainsi s'assurer d'obtenir en bout de ligne un plan d'animation valide et fidèle à la représentation conceptuelle abstraite.

5.2 Représentation de l'information contenue dans la scène statique

Tel qu'expliqué au chapitre 2 dans la problématique de notre projet, ce dernier est effectué en parallèle avec un autre projet inclus dans le projet global GITAN. Celui-ci a pour but de générer, à partir d'une description de scène exprimée dans le format de représentation abstraite présenté à la section 5.1, une scène statique en 3D comportant les objets nécessaires à l'animation de celle-ci. Considérant cela, la génération en parallèle d'un plan d'animation à partir de la même description de scène est un projet indépendant, mais requiert tout de même d'obtenir une petite quantité d'information de la scène statique. Nous présentons donc un format de représentation simple permettant de transmettre cette information d'un projet à l'autre.

5.2.1 Information nécessaire

D'abord, il est nécessaire de définir l'information à représenter par ce format. En effet, toutes les informations présentes dans la scène statique 3D ne sont pas utiles à notre projet. Nous avons donc choisi de créer un format simple de transmission de l'information afin de garder le processus d'analyse le plus efficace possible.

Dans le but de pouvoir générer un plan d'animation à partir d'actions sur les objets de la scène, il est d'abord nécessaire de connaître l'identité de tous les objets présents dans la scène. Certains d'entre eux, souvent les plus importants par rapport au déroulement de la scène,

sont déjà présents de manière conceptuelle dans la description de scène. Par contre, d'autres ont pu être ajoutés dans la scène pour plusieurs raisons dans le processus de génération de la scène statique et ces objets doivent aussi être connus. Une de ces raisons peut être la nécessité de la présence d'un objet pour assurer le fonctionnement d'un autre. Par exemple, pour effectuer l'action de boire de l'eau, celle-ci doit être contenue dans un contenant tel un verre ou une tasse. Bien que ces objets pouvaient ne pas être mentionnés dans la description conceptuelle de la scène, ils sont primordiaux à son déroulement. D'autres raisons peuvent aussi justifier l'ajout d'objets dans la scène, comme l'association courante entre deux objets (exemple : une chaise se place normalement à côté d'une table), ou simplement pour peupler la scène dans un contexte réaliste (exemple : ajouter des meubles pour représenter l'intérieur d'une maison). En conséquence, la liste des objets présents dans la scène statique ne peut être déduite directement de la description conceptuelle de la scène et doit être obtenue du processus de génération de la scène statique.

En plus de connaître l'identité des objets présents dans la scène, une seconde information doit être connue afin de pouvoir connaître suffisamment l'environnement dans lequel le problème de planification sera résolu. Cette information concerne la position spatiale relative de ces mêmes objets. Cela signifie que, sans obtenir explicitement de l'information numérique sur le positionnement des objets, nous devons connaître ce qui fait office de *contrôle* pour chacun de ceux-ci. Le concept de contrôle, initialement défini par l'ontologie *GUM-Space*, précise deux types distincts : le support (*Support*) et le confinement (*Containment*). À quelques exceptions près, tous les objets de la scène doivent posséder un *contrôleur* unique, c'est-à-dire soit un élément servant de support physique (exemple : une table soutenant une assiette) ou un élément pouvant le contenir (exemple : une boîte contenant des jouets). Lorsque le placement spatial des objets est effectué dans le processus de génération de la scène statique, les objets se retrouvent obligatoirement contraints à une relation de contrôle avec un autre élément de la scène. Cette information doit aussi nous être transmise afin de pouvoir connaître l'interaction entre les objets de la scène et ainsi éventuellement en déduire des fonctionnalités.

5.2.2 Format

Le format que nous proposons est à nouveau basé sur le langage XML et comporte une structure très simple puisque l'information à représenter l'est également. Il consiste en une liste des objets présents dans la scène, pour lesquels est spécifié leur contrôleur respectif. L'exemple de la figure 5.5 illustre la représentation d'une boîte et d'une balle dans une scène.

L'élément XML de base pour représenter un objet de la scène est *object*. Chaque objet est représenté par un identifiant unique (*id*) et par sa classe dans notre ontologie des objets du monde (*class*). Cette logique suit celle de la représentation de la scène présentée à la

```
<object>
  <id>ball</id>
  <class>ph:Ball</class>
  <controller>
    <id>box</id>
    <type>gum:Containment</type>
  </controller>
</object>

<object>
  <id>box</id>
  <class>ph:Box</class>
  <controller>
    <id>floor</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>floor</id>
  <class>ph:Floor</class>
</object>
```

Figure 5.5 Représentation XML d'objets de la scène statique

section 5.1. Dans l'exemple présenté, nous pouvons constater la présence de trois objets de types respectifs *Ball*, *Box* et *Floor*. Pour le premier objet (*ball*), le contrôleur spécifié réfère directement au deuxième objet (*box*) par son identifiant. Le type de contrôle est également spécifié comme étant de type *Containment*, ce qui signifie que le premier objet se retrouve confiné par le deuxième objet. Ce deuxième objet (*box*) possède lui aussi un contrôleur référant au troisième objet (*floor*) et indiquant que ce dernier est de type *Support*. Concrètement, l'information obtenue à partir de cette description indique qu'une balle est à l'intérieur d'une boîte, qui se retrouve elle-même sur le plancher. Nous pouvons remarquer que le troisième objet (*floor*) ne possède pas de contrôleur. En effet, le sol étant la base physique de nos scènes, aucun contrôle n'est nécessaire dans ce cas.

Ce format de représentation combine simplicité et efficacité pour la description conceptuelle des objets de la scène statique. Sa similarité avec le format de représentation conceptuelle abstraite est aussi un avantage qui simplifiera le processus d'analyse des informations de la scène pour la formulation du problème de planification. Aussi, par souci de rigueur, un schéma de validation XSD a également été produit pour ce format de représentation. Il est disponible à l'Annexe B.

5.3 Représentation de la connaissance

L'ajout de connaissance dans un environnement virtuel constitue un point crucial de notre projet. Le monde que nous décrivons depuis le début, qui sert d'environnement pour la création de la scène et du plan d'animation de celle-ci, est un monde sémantique. Cela signifie que ses éléments sont représentés par leur nature d'une manière conceptuelle. Bien que des ressources sémantiques externes soient utilisées afin de décrire la scène d'une manière conceptuelle (voir section 5.1), le processus de génération de plan d'animation repose sur une représentation fonctionnelle des objets et actions du monde réel. L'approche que nous proposons pour la représentation de la connaissance implique la conception d'une base de connaissances de type ontologie, jumelée à une description du domaine d'actions possibles du monde réel. Cette section a pour but de présenter ces deux aspects de notre approche.

5.3.1 Conception de l'ontologie

Comme mentionné précédemment dans la revue de littérature sur le sujet (voir section 3.1.2), l'efficacité d'utilisation d'une ontologie réside dans le fait qu'elle doit être spécifique à son domaine d'application. L'ontologie que nous proposons, implémentée en langage OWL, a donc été conçue dans l'optique de répondre aux besoins particuliers de notre projet. Ces besoins se divisent essentiellement en trois considérations différentes ayant été prises en compte lors de la conception de l'ontologie : la représentation des objets, la représentation des contrôleurs et la compatibilité avec le projet parallèle.

Hiérarchie de classes des objets

D'abord, le premier besoin à combler par la conception d'une ontologie est celui de posséder une base de connaissances sur les objets du monde réel. Nous entendons par ces derniers tous les types d'objets physiques pouvant se retrouver dans une scène virtuelle. Puisque les objets concernés sont représentés conceptuellement dans la description initiale de scène (voir section 5.1) mais physiquement dans une scène 3D virtuelle, nous devons posséder de l'information sémantique sur ceux-ci. Avec une hiérarchie de classes suffisamment élaborée, il est possible de représenter les fonctionnalités d'un objet dans un environnement virtuel par l'ensemble des fonctionnalités offertes par ses classes parentes. Ce principe permet ensuite, lors de la conciliation des informations de l'ontologie avec celles du domaine d'actions, de spécifier précisément les catégories d'objets requis par les différentes actions.

Dans le but de présenter un aperçu de la hiérarchie de classes des objets de notre ontologie, la figure 5.6 illustre les classes d'objets implémentées dans celle-ci. Il est à noter que le but de cette ontologie n'est pas d'offrir une hiérarchie de concepts couvrant exhaustivement les objets

du monde réel, mais plutôt de proposer une structure de base avec des concepts concrets. Les objets présents dans notre ontologie ont servi à la définition des actions de notre domaine (voir section 5.3.2) ainsi qu'à l'élaboration et de plusieurs scénarios visant à tester notre solution (voir chapitre 6).

La hiérarchie de classes des objets de l'ontologie est organisée par catégories d'objets selon les fonctionnalités de ceux-ci. Ces catégories dérivent directement de la classe de base *Object* et représentent divers types d'objets présents à l'intérieur d'une maison, puisqu'il s'agit du type d'environnement sur lequel nous nous concentrons pour la démonstration d'application notre projet. Il est intéressant de noter que seules les classes feuilles, c'est-à-dire celles ne possédant pas de sous-classes dans la hiérarchie, représentent des objets concrets pouvant être placés dans une scène virtuelle. Les classes noeuds, celles possédant des sous-classes dans l'arborescence, représentent des catégories d'objets et ne peuvent être instanciées concrètement.

Parmi ces catégories globales, nous retrouvons les classes *Appliance*, *Clothing*, *Dish*, *ElectronicDevice*, *Food*, *Furniture*, *PlumbingFixture* et *Toy*. Plusieurs d'entre elles sont divisées en sous-catégories, permettant de spécifier plus précisément le type des objets en dérivant. Nous retrouvons également des classes concrètes dérivant directement de la classe de base *Object* telles que *Book* et *Human*. Celles-ci peuvent être instanciées au même titre que les autres classes feuilles. Elle n'ont simplement pas été classifiées comme les autres puisque les scénarios de test ne le nécessitaient pas. Ceci dit, presque toutes les classes de l'ontologie pourraient être relocalisées ou classifiées différemment sans affecter le fonctionnement des processus qui utilisent la base de connaissances.

Afin d'analyser plus en détails la classification des objets, prenons l'exemple de la classe *PlumbingFixture*, montrée dans le coin inférieur droit de la figure 5.6. Cette classe a pour but de représenter tout type d'équipement de plomberie pouvant être présents dans un bâtiment. Elle possède sept classes concrètes : *Bath*, *Shower*, *Toilet*, *Urinal*, *Sink*, *DrinkingFountain* et *Tap*. En plus d'être catégorisées comme équipement de plomberie, ces classes concrètes sont aussi classifiées selon leurs fonctions respectives. Par exemple, la classe *BathingFixture* représente les équipements de plomberie permettant de se laver et regroupe les classes concrètes *Bath* et *Shower*. Toujours dans la même catégorie, la classe *WaterSource*, représentant un équipement pouvant fournir de l'eau, regroupe les classes concrètes *DrinkingFountain* et *Tap*. L'importance de ces catégories est expliqué plus en détails lors de la présentation du domaine d'actions (voir section 5.3.2), car les fonctionnalités précises des classes objets y sont exploitées.

Il est à noter que la classe *Container*, qui fait partie de la hiérarchie de classe des objets présentée dans la figure 5.6 mais sans être explicitée, est aussi une sous-classe de la classe de

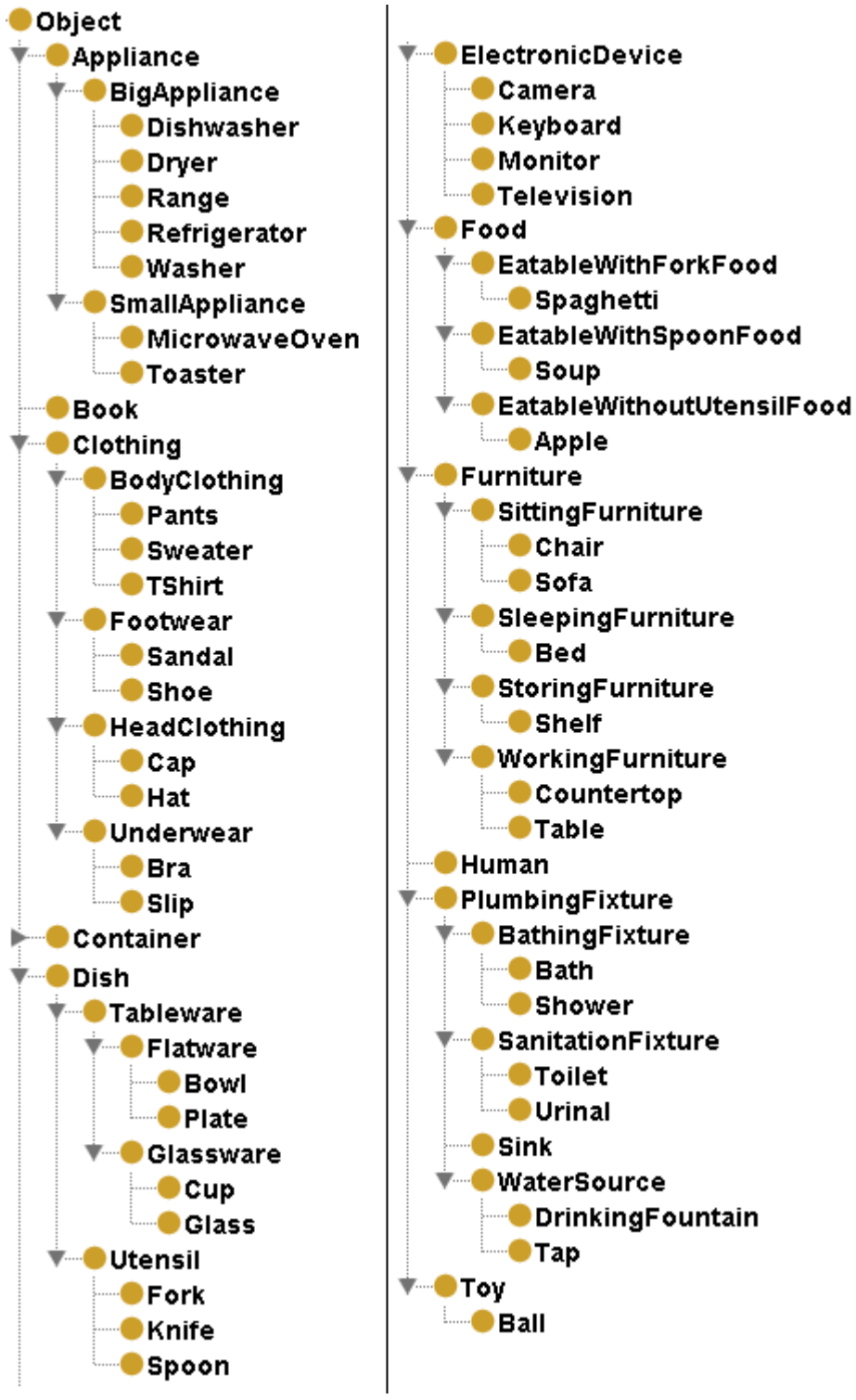

Figure 5.6 Hiérarchie de classes des objets de l'ontologie

base *Object*. Dû à son rôle plus important dans la base de connaissances, elle sera présentée plus en détails à la section suivante.

Figure 5.7 Hiérarchie de la classe Liquid

La figure 5.7 montre la classe *Liquid* de notre ontologie. Cette classe n'est pas une sous-classe de la classe *Object* car nous avons décidé de la traiter différemment. En effet, dans les scènes virtuelles, les liquides ne peuvent être traités comme un objet entier puisqu'ils peuvent être présent à plusieurs endroits dans la scène, se diviser, se combiner ou provenir d'une source non-liquide (exemple : de l'eau provenant d'un robinet). Nous avons donc prévu une classe spéciale *Liquid* pour ces cas et ainsi permettre des actions spécifiques à ceux-ci (voir section 5.3.2) ainsi qu'un type de contenant particulier (voir section suivante). Pour l'instant, la seule classe concrète dérivant de *Liquid* à avoir été incluse dans notre ontologie est *Water*, mais il y a bien sûr la possibilité d'en ajouter d'autres et de complexifier la classification.

Hiérarchie de classes des contrôleurs

Tel qu'expliqué précédemment à la section 5.2.1, le concept de contrôle est très important dans notre représentation de la scène car il indique l'essentiel des interactions spatiales entre les objets de la scène. Celui-ci est donc aussi présent dans notre ontologie, sous forme d'objets de type *Controller*, se divisant en deux classes distinctes : *Container* et *Support*. La figure 5.8 illustre la hiérarchie de classes implémentée dans l'ontologie pour représenter le type *Container*.

D'abord, nous pouvons constater que la classe *Container* dérive directement de la classe de base *Controller* définie dans l'ontologie. Comme mentionné dans la section précédente, la classe *Container* dérive également de la classe de base *Object*. Nous pouvons donc résumer l'utilité des objets dérivant de la classe *Container* par des objets de la scène pouvant offrir un confinement à d'autres objets. Plusieurs sous-classes sont définies pour représenter divers types de contenants : *Appliance*, *Box*, *Flatware*, *HumanContainer* et *LiquidContainer*. Ces sous-classes sont parfois elles-mêmes subdivisées afin d'offrir une meilleure granularité pour les types génériques tels que *LiquidContainer*, par exemple. Tous ces types de contenants n'ont pas la même utilité dans une scène virtuelle, d'où la nécessité de les définir précisément. Les diverses classes d'objets de la scène présentées à la section précédente spécifient un type de contenant précis, ce qui permet à la scène d'avoir une certaine cohérence. Les relations permettant de spécifier une telle contrainte sont présentées à la section suivante.

Afin de mieux comprendre le rôle des sous-classes de la classe *Container*, examinons les

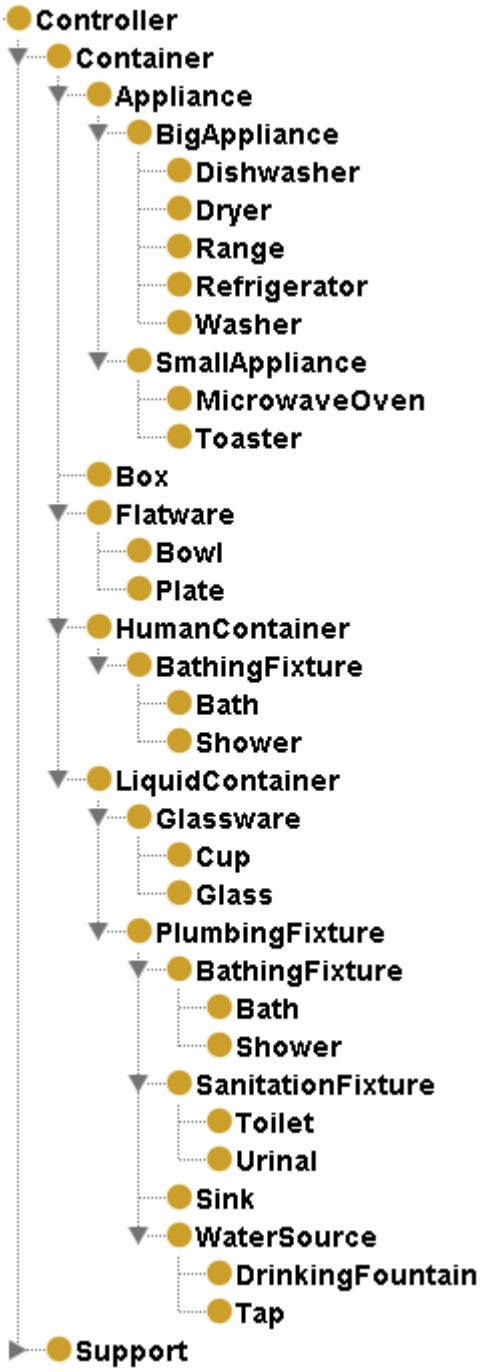

Figure 5.8 Hiérarchie de la classe Container

classes *HumanContainer* et *LiquidContainer*. La classe *HumanContainer*, comme son nom l'indique, représente les objets pouvant contenir un humain. Parmi les classes de l'ontologie que nous avons définies, seule la classe *BathingFixture*, incluant les classes concrètes *Bath* et *Shower*, correspond à cette description. Ceci signifie qu'une action du domaine (voir section

5.3.2) qui impliquerait à l'acteur de type *Human* d'entrer à l'intérieur d'un objet de la scène obligerait que cet objet soit de type *BathingFixture*. De son côté, la classe *LiquidContainer* représente les objets pouvant contenir un liquide. Cette distinction est importante puisque, comme expliqué dans la section précédente, la classe *Liquid* et la classe *Object* de notre ontologie sont des classes de base distinctes. Les instances de la classe *Liquid* et de ses sous-classes ont donc *LiquidContainer* comme seul type de contenant possible. Deux classes de l'ontologie sont spécifiés comme étant également des sous-classes de *LiquidContainer* : *Glassware* et *PlumbingFixture*. Dans le cas de la classe *Glassware* et de ses classes concrètes *Cup* et *Glass*, elles sont définies pour contenir uniquement des liquides de type *DrinkableLiquid*. La contrainte appliquée à la classe *PlumbingFixture* et à ses classes dérivées est encore plus spécifique car elles peuvent contenir uniquement un liquide de type *Water*.

Ces types de contenants définis ainsi que leur contraintes spécifiques, implémentées sous forme d'axiomes servant à la définition des classes de l'ontologie, permettent de garder une cohérence dans l'élaboration du plan d'animation, puisque les objets sont contraints à jouer majoritairement le rôle qu'ils occupent dans le monde réel. Notre modèle ne permettant pas d'identifier un usage régulier par rapport à un usage non conventionnel (exemple : utilisation d'un bol pour boire de l'eau), la restriction à l'usage régulier constitue le meilleur choix pour assurer la cohérence des scènes virtuelles et de leur plan d'animation.

Le deuxième type de contrôleur présent dans notre ontologie est représenté par la classe *Support*. La figure 5.9 illustre la hiérarchie de classes implémentée dans l'ontologie pour représenter ce type.

La première observation pouvant être effectuée sur le type *Support* par rapport au type *Container* présenté ci-haut est qu'il ne dérive pas directement de la classe de base *Object*. Cela ne signifie pas qu'aucun objet concret de la scène ne peut servir de support, mais plutôt que tous les supports ne sont pas nécessairement des objets de la scène dérivant de la classe *Object*. Contrairement à la classe *Container* où tous les sous-types spécifient leurs propres contraintes et sont tous regroupés sous le même type de base, la classe Support comporte trois sous-classes distinctes, chacune représentant un type de support physique différent. Ces types sont les suivants :

AboveHorizontalSupport Ce type de support, comme son nom l'indique, indique que le support est effectué de façon horizontale au-dessus de l'objet concerné. L'application la plus commune de ce type de support est un plafond pouvant supporter, par exemple, un luminaire plafonnier, un détecteur de fumée ou encore un ventilateur. Ce type peut être appliqué dans le cas d'une scène statique, mais rares sont les interactions avec ces objets dans une scène animée. Aucune sous-classe d'objets de la scène n'a donc été ajoutée à cette classe dans l'ontologie puisque les scénarios ne le nécessitent pas, mais

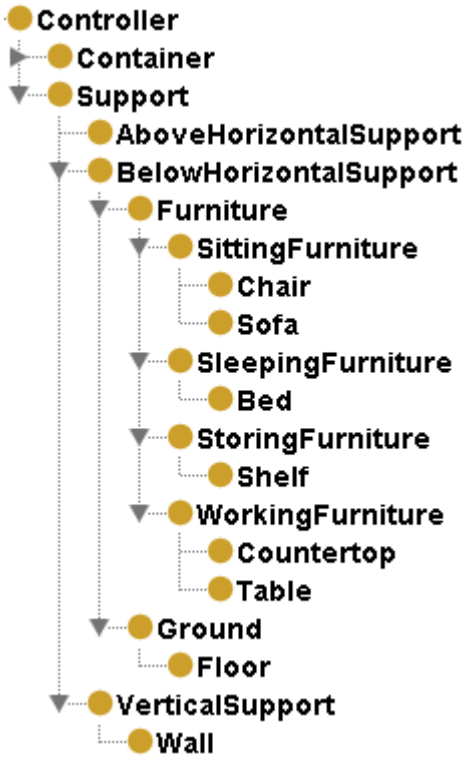

Figure 5.9 Hiérarchie de la classe Support

la catégorie existe tout de même.

BelowHorizontalSupport Ce type de support indique également que le support est effectué de façon horizontale, mais qu'il s'applique sous l'objet concerné. Il est celui le plus souvent utilisé et représenté dans les scènes virtuelles pour la simple raison qu'il s'agit d'une application directe de la gravité et ne nécessite pas d'artéfact précis pour assurer le support, contrairement aux deux autres types. La majorité des objets dérivant de la classe *Object* et servant de support se retrouvent dans cette catégorie (exemples : table, bureau, chaise, etc.), ainsi que le plancher.

VerticalSupport Ce type de support indique que le support est effectué de façon verticale par rapport à l'objet concerné. La majorité des applications de ce type de support concernent un mur, où peuvent y être supportés divers types d'objets (exemples : cadre, fenêtre, horloge, etc.).

Tel qu'expliqué précédemment à la section 5.2.1, le support est un composante essentielle de la scène virtuelle, dans le sens où chaque objet de la scène doit absolument avoir un support physique. La raison pour laquelle les éléments de la pièce tels que le plancher, le plafond et les murs ne sont pas considérés comme des objets est que nous considérons qu'ils constituent la base de la scène virtuelle. Puisque notre projet se concentre sur la génération de scène

animées dans un environnement à une seule échelle de grandeur (exemple : l'intérieur d'une pièce), il n'est pas nécessaire de spécifier plus d'information que nécessaire quant au contrôle appliqué sur ces éléments de la pièce. Dans les scénarios de test, lorsqu'un objet de la scène est supporté par un de ces éléments, il s'agit de la dernière relation de contrôle de la chaîne (exemple : un livre supporté par un bureau, lui-même supporté par le plancher).

Afin d'illustrer les différences entre certaines classes dérivant de la classe *Support*, prenons l'exemple de la classe *Furniture*, présentée à la figure 5.9. Cette classe est divisée en quatre sous-classes différentes : *SittingFurniture*, *SleepingFurniture*, *StoringFurniture* et *Working-Furniture*. Bien que ces quatre sous-classes constituent toutes des classes dérivées de *Below-HorizontalSupport*, donc pouvant supporter un objet horizontalement, elles se distinguent les unes des autres par leur fonctionnalité dans une interaction avec l'acteur. Ainsi, bien que les objets de ces quatre classes peuvent physiquement supporter un human, seuls des objets de type *Chair* ou *Sofa*, donc dérivant de *SittingFurniture*, permettent à l'acteur de s'y asseoir ou de s'y tenir en position assise. L'action *sit-down* définie dans le domaine d'actions (voir section 5.3.2) accepte donc uniquement un paramètre de type *SittingFurniture*, restreignant ainsi la possibilité que l'acteur puisse s'asseoir sur un autre type d'objet non prévu à cet effet. Ces types de contraintes, combinées à celles du domaine d'actions, permettent de limiter les objets de type support à leur usage régulier, même si la hiérarchie de classes de l'ontologie seule leur confère davantage de fonctionnalités possibles.

Compatibilité avec la génération de scène statique

Les deux sections précédentes exposent les composantes de l'ontologie utilisées dans notre projet comme représentation partielle de la connaissance. Comme mentionné précédemment, notre ontologie a été conçue dans l'optique de servir de base de connaissances pour deux projets, le nôtre ainsi qu'un second projet parallèle de génération de scènes virtuelles statiques. Certaines décisions de conception et d'implémentation ont donc été prises en considérant l'apport aux deux projets. Nous présentons ici un aperçu de trois composantes de l'ontologie qui ne sont pas directement reliées au processus de génération de plan d'animation, mais qui ajoutent de l'information nécessaire à la génération de scène statique et enrichissent la base de connaissances.

Premièrement, l'ontologie prévoit quatre classes de base indépendantes pour représenter différents éléments de connaissances. Dans les deux sections précédentes, nous avons vu la hiérarchie de classes des trois premières classes de base : *Object*, *Liquid* et *Controller*. La quatrième classe de cet ensemble est la classe *Process*, qui représente les actions du domaine. Bien que, pour les besoins de notre projet, ces actions sont représentées sous un autre format que dans l'ontologie (voir section 5.3.2), nous les avons incluses dans le but de faire un lien

avec la représentation conceptuelle abstraite de la scène. Le but d'avoir une représentation des actions dans l'ontologie au niveau de la génération de scène est de connaître les éléments et objets nécessaire à l'exécution de cette action. Prenons comme exemple l'action de boire, exécutée par un acteur dans une scène virtuelle. Afin que l'acteur puisse effectuer cette action, il est obligatoire que la scène contienne un liquide pouvant être bu (*DrinkableLiquid*). De plus, il est nécessaire de fournir à l'acteur le ou les outils nécessaires à l'exécution de l'action. Dans notre cas, un objet de type *Glassware* est nécessaire pour contenir le liquide afin qu'il soit bu. Ces informations sont donc associées à l'action de boire dans l'ontologie et répertoriés sous la classe *Process*.

En plus des hiérarchies de classes, l'avantage d'utiliser une ontologie comme base de connaissances est qu'il est possible de définir des *propriétés*, c'est-à-dire des relations entre les classes. Afin de représenter plusieurs des contraintes expliquées dans la section 5.3.1, nous avons défini des propriétés permettant d'appliquer ces contraintes sur les différentes classes de l'ontologie. La figure 5.10 illustre la liste des propriétés définies.

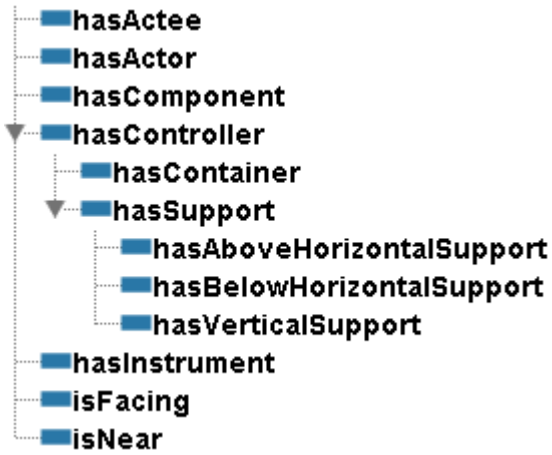

Figure 5.10 Propriétés définies dans l'ontologie

Sans entrer dans les détails de conception puisque ces propriétés concernent majoritairement la génération de la scène statique, ces propriétés sont définies pour des usages précis sur les classes de l'ontologie. Par exemple, les propriétés *hasContainer* ainsi que *hasSupport* et ses sous-propriétés ont comme domaine d'application les objets et liquides de l'ontologie, auxquels y sont associés un contenant ou un support obligatoire, selon le cas. D'un autre côté, les propriétés *hasActee*, *hasActor* et *hasInstrument* ont comme domaine d'application les membres de la classe de base *Process* présentée ci-haut, auxquels sont associés les éléments essentiels à l'exécution de l'action concernée. Par exemple, la figure 5.11 illustre les propriétés spécifiées pour la classe *Drink*.

Figure 5.11 Propriétés appliquées à la classe *Drink*

Les propriétés *hasActee*, *hasActor* et *hasInstrument* sont alors utilisées pour spécifier que l'action de boire nécessite un acteur humain, un liquide pouvant être bu ainsi qu'un contenant de liquide permettant de boire. Une scène abstraite représentant l'action de boire comporterait donc obligatoirement ces éléments et ainsi assurerait en partie la faisabilité de la génération d'un plan d'animation.

Enfin, une troisième composante de l'ontologie est exploitée afin de représenter de l'information sur les objets de la scène. Il s'agit de l'usage des attributs, c'est-à-dire l'ajout d'informations numériques aux objets conceptuels de la base de connaissances. Notre ontologie prévoit quatre propriétés quantitatives associées à chacune des classes d'objets : la masse ainsi que les dimensions selon trois axes. Ces propriétés quantitatives sont utiles pour le placement spatial lors de la génération de la scène statique, mais ont peu d'incidence sur notre projet.

5.3.2 Domaine d'actions du monde réel

La seconde partie de notre approche de représentation de la connaissance concerne les actions du monde réel. Dans le but de générer un plan d'animation pour une scène virtuelle, il est nécessaire d'identifier le domaine des possibilités pour l'acteur. Dans le formalisme de la planification classique (voir section 4.2.3), ces transitions d'états portent également le nom d'*actions*. La représentation sémantique de ce domaine demande une modélisation réaliste des actions du monde réel. Il est donc nécessaire de définir un ensemble de prédicats pouvant constituer les états de notre monde ainsi qu'un ensemble d'actions représentant les transitions entre ces états. Cette section a pour but de présenter ces deux ensembles.

Contrairement à la représentation des objets du monde, pour lesquels nous avons choisi un format indépendant de représentation de la connaissance (une ontologie en langage OWL) avant de l'utiliser dans d'autres parties de notre programme, le domaine d'actions du monde réel que nous avons défini est représenté directement en langage PDDL (Planning Domain

Definition Language). La structure particulièrement stricte de ce langage de formulation de problèmes de planification, combinée au fait qu'il ne nous était pas nécessaire d'utiliser cette connaissance ailleurs que pendant la résolution du problème de planification, a éliminé le besoin de sélectionner un autre format de représentation. Nous avons donc choisi d'utiliser directement une formulation de l'information en langage PDDL, évitant du même coup la complexité de représenter ces informations précises dans un format moins adapté à la situation, OWL par exemple. La possibilité d'utiliser un format de représentation plus abstrait est disuctée dans les travaux futurs, à la section 8.2.

Définition des prédicats

Tel qu'expliqué à la section 4.2.2, l'ensemble des prédicats logiques définis pour un problème de planification définissent le domaine de ce problème. Les états possibles lors de la résolution du problème sont représentés par des conjonctions de prédicats. Il est donc nécessaire de définir les prédicats permettant d'exprimer en termes logiques le déroulement de l'animation dans la scène virtuelle. Nous considérons trois types d'information à représenter : la position de l'acteur, les relations de contrôles entre les éléments de la scène et les indicateurs d'actions.

Premièrement, la position de l'acteur est une information cruciale à représenter pour assurer un déroulement cohérent des actions de la scène. En effet, plusieurs interactions avec les objets de la scène nécessitent que l'acteur ait accès aux objets concernés. Il est donc nécessaire de définir des prédicats permettant de représenter la position de l'acteur relativement aux objets de la scène. Par contre, cette position ne peut être exprimée explicitement car la représentation des scènes dans notre projet est effectuée en faisant abstraction des données numériques. La solution que nous proposons est donc d'identifier la position de l'acteur comme étant celle de l'objet de la scène concerné par l'action en cours. Nous utilisons la préposition *at* pour indiquer que l'acteur humain est à la même position qu'un objet de la scène. En pratique, dans une scène 3D, l'acteur ne serait évidemment pas à la position exacte de l'objet, mais nous faisons abstraction des mécanismes d'interaction propres à chaque objet et représentons la position de l'acteur par celle de l'objet. Les deux prédicats résultants sont présentés au tableau 5.1.

Tableau 5.1 Prédicats de position de l'acteur

at ?human ?object
at-start ?human

Chaque prédicat PDDL peut être décomposé en deux parties : le nom et les paramètres. Le

premier prédicat présenté ci-haut a comme nom *at* et possède deux paramètres indiqués par le symbole ?, *human* et *object*. Le nom des paramètres est arbitraire, mais nous nous sommes fixés comme convention de les nommer selon les types auxquels nous voulions restreindre les prédicats. Par exemple, le prédicat *at* doit être appliqué à un acteur de type *Human* et à un objet de type *Object*.

Les deux prédicats présentés ci-haut ont donc comme fonction de représenter la position de l'acteur dans une scène virtuelle sans données numériques. Le prédicat *at-start* sert à représenter la position de départ de l'acteur, la seule situation où il ne peut se situer à la même position qu'un objet. Une fois un premier déplacement effectué par l'acteur, ce prédicat disparaît de l'état courant et la position relative de l'acteur est uniquement représentée par le prédicat *at*.

Deuxièmement, les relations de contrôle entre les éléments de la scène doivent également être représentées. Elles se retrouvent d'abord dans la configuration initiale de la scène (voir section 5.2), puis évoluent selon les différents déplacements et actions effectués au cours de l'animation. Les relations de confinement (*Containment*) et de support (*Support*), issues de *GUM-Space*, sont bien sûr représentées. De plus, nous avons identifié une troisième relation, celle de préhension, où un objet de la scène se retrouve temporairement sans support ni confinement puisqu'il est saisi et tenu par un acteur humain. Les prédicats résultants sont sont présentés au tableau 5.2.

Tableau 5.2 Prédicats de relations de contrôle

containing ?container ?object
containing ?liquidContainer ?liquid
supporting ?support ?object
holding ?human ?object

La relation de confinement (*containing*) est divisée en deux prédicats à cause de la distinction entre les classes *Object* et *Liquid* dans l'ontologie (voir section 5.3.1), mais représente la même relation de contrôle dans les deux cas. Pour leur part, les relations de support (*supporting*) et de préhension (*holding*) ne peuvent impliquer directement des liquides alors une seule définition suffit.

Troisièmement, un autre type de prédicats présent dans notre domaine concerne directement l'exécution des actions dans la scène. Puisque notre projet est surtout axé sur le déroulement de la scène plutôt que sur sa finalité, nous avons défini un ensemble de prédicats ayant pour rôle d'indiquer les actions se déroulant dans la scène. Cet ensemble n'est bien sûr pas exhaustif puisqu'il est directement lié à la définition des actions du domaine (voir section suivante). Par contre, il est suffisamment grand pour couvrir les scénarios de test que nous

avons définis pour illustrer notre preuve de concept (voir chapitre 6).

Les prédicats indicateurs d'actions sont divisés en deux sous-types : les actions en cours et les actions effectuées. L'ensemble des prédicats indiquant des actions en cours est restreint puisque la majorité des actions du domaine sont définies comme des événements ponctuels. Cette simplification est possible puisque l'environnement des problèmes de planification est discret. Cependant, nous avons ciblé quelques actions parmi celles définies pouvant être identifiées comme non ponctuelles et donc, à un certain point lors de la résolution, pouvant être représentées comme « en cours ». Les prédicats du tableau 5.3 ont été définis pour indiquer cet état.

Tableau 5.3 Prédicats d'action en cours

is-moving ?human
is-drinking ?human
is-eating ?human
is-washing-itself ?human
is-washing ?human ?object
is-aiming ?human ?object ?object
is-aiming ?human ?object ?support
action-mutex-enabled

Il s'agit là de six actions que nous avons définies comme pouvant être dans l'état « en cours ». Ce type d'actions de mouvement nécessite une définition plus complexe des actions du domaine, ou transitions (voir section suivante), car nous devons spécifier des transitions atomiques distinctes pour le début et la fin du cycle de chacune de ces actions afin que leur déroulement ne soit pas ponctuel dans le processus de résolution du problème. Le dernier prédicat, *action-mutex-enabled*, est utilisé pour éviter que l'acteur puisse entreprendre une autre action non désirée pendant l'exécution d'une action non ponctuelle. Par exemple, lorsque l'acteur exécute l'action de se laver, il ne peut entreprendre de manger sans au préalable mettre fin à son action de se laver, ce qui serait possible si un tel mécanisme n'était pas présent. Plus de détails sur ce dernier sont donnés à la section suivante. Enfin, la double définition du prédicat *is-aiming* est encore une fois dû à la distinction entre les classes *Object* et *Support* dans l'ontologie, alors qu'il est possible de viser vers les deux.

Finalement, les prédicats présents en plus grand nombre dans le domaine sont des indicateurs d'action effectuée. Chaque action définie dans le domaine possède au minimum un indicateur permettant de signifier que l'action a été effectuée dans la scène. L'existence de ces derniers est justifiée par le fait que dans le contexte de notre projet, l'exécution d'une action est parfois, voire souvent, le but principal de la scène à animer. Il est donc nécessaire de pouvoir formaliser clairement l'exécution de ces actions dans le problème de planification.

Bien que la liste des actions du domaine ne soit présentée qu'à la section suivante, nous présentons ici la liste des prédicats possibles du domaine, résultant de l'exécution de ces actions. La liste de ces prédicats est présentée au tableau 5.4.

Tableau 5.4 Prédicats d'action effectuée

has-moved ?human
has-moved-to ?human ?object
has-drunk ?human ?drinkableLiquid
has-eaten ?human ?food
has-ingested ?human
has-washed-itself ?human
has-washed ?human ?object
has-aimed ?human
has-aimed-at ?human ?object ?object
has-taken ?human ?object
has-taken-from ?human ?object ?support
has-taken-from ?human ?object ?container
has-put ?human ?object
has-put-on ?human ?object ?support
has-placed-in ?human ?object ?container
has-sat-down ?human
has-sat-down-on ?human ?sittingFurniture
has-lied-down ?human
has-lied-down-on ?human ?sleepingFurniture
has-stood-up ?human
has-stood-up-from ?human ?sittingFurniture
has-stood-up-from ?human ?sleepingFurniture
has-changed-posture ?human
has-pushed ?human ?object
has-pushed-to ?human ?object ?object
has-thrown ?human ?object
has-thrown-at ?human ?object ?object
has-thrown-at ?human ?object ?support
has-thrown-in ?human ?object ?container
has-poured ?human ?water
has-poured-from ?human ?water ?waterSource
has-poured-in ?human ?water ?liquidContainer

Cette liste conclut l'ensemble des prédicats disponibles pour le domaine que nous avons défini dans le cadre de ce projet. Chaque état initial, but ou état temporaire des problèmes de planification formulés (voir section 5.4) est représenté par une conjonction de ces prédicats.

Définition des actions

Le point central du domaine que nous avons défini dans le cadre de notre projet est la définition des actions. Chaque plan généré en bout de ligne lors de la phase de résolution du problème de planification est obligatoirement une liste d'actions appliquées sur les éléments de la scène virtuelle. L'ensemble des actions doit donc refléter toutes les possibilités d'interaction de l'acteur dans la scène virtuelle. Comme mentionné précédemment à quelques reprises, notre but dans la définition du domaine n'est pas de couvrir exhaustivement les actions du monde réel, mais plutôt de prouver la faisabililité de cette approche de conceptualisation du monde réel. Nous présentons donc ici le formalisme utilisé pour la définition des actions ainsi que quelques exemples concrets.

Les actions du domaine sont définies en langage PDDL, reconnaissant le typage des paramètres et suivant un formalisme STRIPS (voir section 4.2.2). Afin de présenter concrètement le formalisme utilisé pour la définition de nos actions, examinons l'exemple de la figure 5.12, qui représente la définition de l'action *take*, en PDDL.

```
(:action take
  :parameters (
    ?human - Human
    ?object - Object
    ?container - Container
    )
  :precondition (and
    (at ?human ?object)
    (containing ?container ?object)
    (not (action-mutex-enabled))
    )
  :effect (and
    (holding ?human ?object)
    (not (containing ?container ?object))
    (has-taken ?human ?object)
    (has-taken-from ?human ?object ?container)
    )
)
```

Figure 5.12 Définition de l'action *take* en PDDL

Les mots-clés PDDL dans la définition d'une action sont les mots *action*, *parameters*, *precondition*, *effect* et l'opérateur *and*. La première ligne spécifie le nom de l'action à définir, dans ce cas-ci *take*. Le nom des actions n'est pas nécessairement unique, donc une action peut posséder plusieurs définitions différentes. Dans notre cas, il s'agit de l'action de prendre un objet initialement confiné dans un contenant. Les paramètres concernés sont l'acteur, de type

Human, l'objet à prendre, de type *Object* et le contenant, de type *Container*. De la même façon que pour la définition des prédicats, les paramètres sont identifiés par le symbole ? suivi du nom de la variable, tandis que les types sont identifiés par le nom de la classe débutant par une lettre majuscule. Suivent ensuite la liste des préconditions et la liste des effets. Par l'utilisation de l'opérateur *and*, il est possible de déterminer que ces listes constituent une conjonction de plusieurs prédicats. Dans le cas de notre version de l'action *take*, les préconditions stipulent que l'acteur doit être à la même position que l'objet à prendre (prédicat *at*) et que l'objet doit être initialement confiné dans le contenant (prédicat *containing*). Le prédicat *action-mutex-enabled* doit également être désactivé, signifiant qu'une action non ponctuelle n'est pas déjà en cours d'exécution. Lorsque ces conditions sont respectées, l'action *take* peut être appliquée sur l'état courant, ce qui résulte à l'ajout des prédicats de la liste d'effets dans celui-ci. D'abord, l'ajout du prédicat *holding* combiné à la suppression du prédicat *containing* signifient que le contrôle de l'objet est désormais de type préhension et assuré par l'acteur plutôt que par le contenant. Ensuite, les deux prédicats indicateurs d'action effectuée *has-taken* et *has-taken-from* sont ajoutés à l'état courant avec les paramètres concernés.

Pour la réalisation de nos scénarios de test, nous avons défini un domaine d'actions suffisamment grand pour couvrir diverses actions de base réalisables dans un contexte de scène virtuelle animée. La liste de base des actions implémentées est la suivante : *take*, *put on*, *place in*, *get in*, *get out*, *get on get off*, *sit down*, *lie down*, *stand up*, *push*, *throw*, *throw at*, *throw in*, *pour*, *drink*, *eat*, *wash itself*, *wash*, *aim* et *move*. Certaines des ces actions ont plusieurs définitions pour répondre à des contextes d'applications différents, tandis que d'autres voient leur définition divisée en plusieurs actions pour répondre au besoin de non ponctualité expliqué à la section précédente. La définition appropriée de l'action qui sera choisie dans chaque contexte spécifique peut être définie à l'avance par les attributs spécifiés dans la description de scène, ou plutôt décidée à l'exécution par évaluation des possibilités offertes par l'état courant de la scène. La liste complète des actions concrètes définies est disponible à l'annexe C.

5.4 Formulation d'un problème de planification

La dernière étape du processus de génération d'un plan d'animation est la formulation d'un problème de planification résoluble à partir des données obtenues auprès diverses sources d'information présentées aux sections 5.1, 5.2 et 5.3. Les informations contenues dans les fichiers XML de description de scène combinées à celles de la base de connaissances et du domaine d'actions du monde réel constituent l'ensemble des données à traiter. Ce traitement constitue en une modélisation de la scène à partir de ces informations, à partir de laquelle

sont définis l'environnement, les prédicats initiaux et les prédicats de but d'un problème de planification classique. Ce dernier est ensuite résolu à l'aide d'un planificateur.

Cette section a pour but de présenter le processus de génération du plan d'animation, décomposé en ses étapes principales. Afin d'en faciliter la compréhension, un exemple détaillé accompagnera la description de toutes les étapes. La figure 5.13 présente un résumé schématique de ce processus.

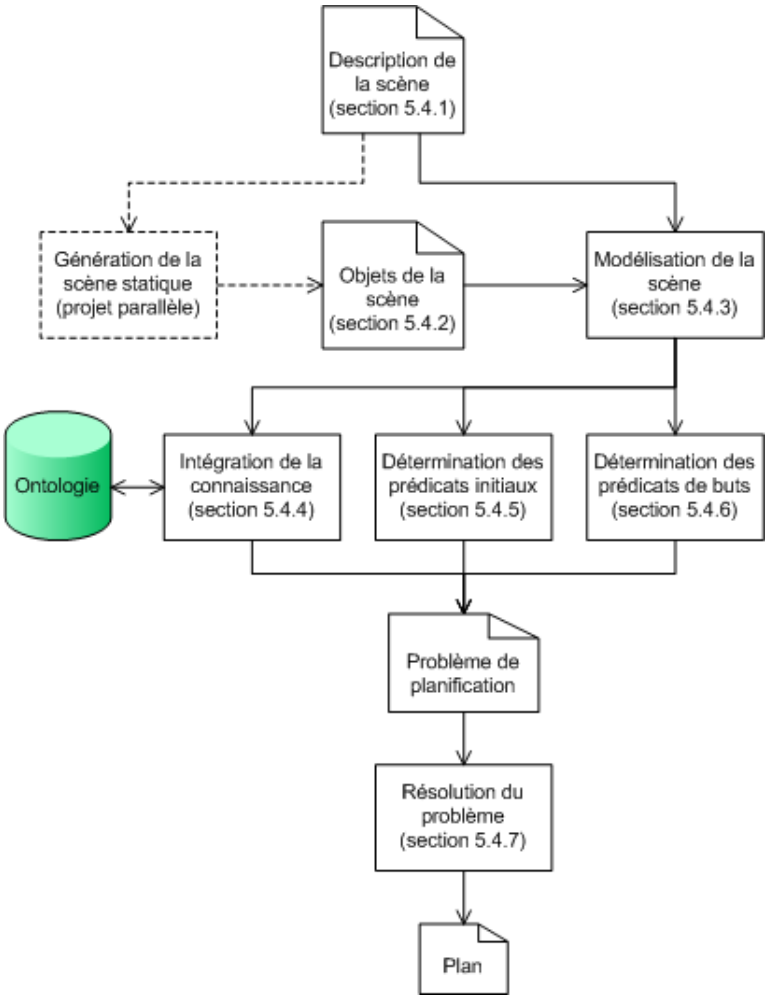

Figure 5.13 Processus de génération du plan d'animation

5.4.1 Description de la scène animée

Le point de départ du processus entier de génération d'un plan d'animation est d'avoir une situation à représenter. Bien sûr, comme notre base de connaissances et notre domaine d'actions ne couvrent pas exhaustivement le monde réel, nous devons nous assurer que la situation que nous voulons représenter comporte uniquement des éléments connus. Consi-

dérant cela, toute situation cohérente et pouvant être exprimée à l'aide de notre format de représentation conceptuelle abstraite est représentable.

Nous analyserons un exemple comportant l'action de manger, un endroit précis où effectuer l'action ainsi qu'une modalité spatiale définie par rapport à cet endroit. La représentation textuelle exacte de la situation est « She ate her spaghetti on the sofa ». Cette dernière est spécifiée uniquement par souci de compréhension puisqu'aucun élément textuel n'intervient dans le processus. Par contre, dans le contexte du projet GITAN (voir chapitre 2), il est correct de supposer que la représentation initiale de la situation serait textuelle, pour ensuite être analysée sémantiquement et transformée en une représentation conceptuelle abstraite. La figure 5.14 illustre la situation proposée dans le format de représentation présenté à la section 5.1.

```
<sceneDeclaration>
  <configuration xsi:type="AffectingSimpleMotion">
    <process>fn:Ingestion</process>
    <actor>
      <name>She</name>
      <class>ph:Human</class>
    </actor>
    <actee>
      <name>spaghetti</name>
      <class>ph:Spaghetti</class>
    </actee>
    <placement>
      <relatum>
        <name>sofa</name>
        <class>ph:Sofa</class>
      </relatum>
      <spatialModality>
        <type>gum:Support</type>
      </spatialModality>
    </placement>
  </configuration>
</sceneDeclaration>
```

Figure 5.14 Représentation de la situation « She ate her spaghetti on the sofa ».

Il s'agit donc d'une action de type *AffectingSimpleMotion* correspondant au *process Ingestion* défini par *FrameNet*. L'acteur est de type *Human* et l'élément affecté par l'action de type *Spaghetti*. L'action possède un emplacement précis, de type *Sofa*, auquel une modalité spatiale de support est rattachée. Cette description de scène est cohérente, fidèle à la situation à représenter et respecte la sémantique de représentation de notre format (voir section 5.1).

5.4.2 Obtention des objets de la scène

Une fois la description de scène obtenue, une scène statique doit être générée à partir de celle-ci tel qu'expliqué à la section 5.2. Supposons une scène standard comprenant une douzaine d'objets habituellement présents à l'intérieur d'une pièce de maison, par exemple une salle à manger. Bien sûr, comme mentionné précédemment, la scène statique générée doit minimalement contenir les objets explicitement présents dans la description de scène de la section 5.4.1. Cependant, plusieurs autres objets peuvent y être ajoutés pour ajouter au réalisme de la scène, ce qui est le cas pour l'exemple présent. La figure 5.15 montre la description des objets présents dans la scène statique.

La scène statique comporte donc douze objets en plus du plancher de type *Floor*. Ce dernier sert d'abord de support pour un acteur humain, une table, une chaise ainsi qu'un sofa. Ensuite, plusieurs objets sont déposés sur la table : une assiette, un bol, un verre, une fourchette, une cuillère et un couteau. Enfin, l'assiette contient du spaghetti, tandis que le bol contient une soupe.

5.4.3 Modélisation de la scène

À partir de l'information contenue dans les descriptions XML montrées dans les figures 5.14 et 5.15, la scène doit être modélisée afin de pouvoir en faire le traitement. Cette modélisation consiste à représenter toutes les informations de la scène animée de façon spécifique à leur type. Trois types d'informations sont définis, auxquels nous associerons les éléments concernés de l'exemple présent. Ces types sont les objets, les relations de contrôle et l'action.

Objets

Les objets de la scène sont représentés à leur plus simple expression, c'est-à-dire uniquement à l'aide de leur identificateur et de leur classe. La liste des objets modélisés pour notre exemple est présentée au tableau 5.5.

Relations de contrôle

Chacune des relations de contrôle est associée à deux éléments de la scène, le contrôleur et l'objet contrôlé, ainsi qu'au type de contrôle correspondant. Comme expliqué précédemment à la section 5.2.1, les deux types de contrôle possibles sont *Support* et *Containment*. Les relations sont modélisées par une association aux objets de la scène déjà spécifiés à la section précédente par le biais de leur identifiant (ID). La liste des relations modélisées pour notre exemple est présentée au tableau 5.6.

```
<object>
  <id>She</id>
  <class>ph:Human</class>
  <controller>
    <id>floor</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>spaghetti</id>
  <class>ph:Spaghetti</class>
  <controller>
    <id>plate</id>
    <type>gum:Containment</type>
  </controller>
</object>

<object>
  <id>plate</id>
  <class>ph:Plate</class>
  <controller>
    <id>table</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>soup</id>
  <class>ph:Soup</class>
  <controller>
    <id>bowl</id>
    <type>gum:Containment</type>
  </controller>
</object>

<object>
  <id>bowl</id>
  <class>ph:Bowl</class>
  <controller>
    <id>table</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>glass</id>
  <class>ph:Glass</class>
  <controller>
    <id>table</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>spoon</id>
  <class>ph:Spoon</class>
  <controller>
    <id>table</id>
    <type>gum:Support</type>
  </controller>
</object>
```

```
<object>
  <id>fork</id>
  <class>ph:Fork</class>
  <controller>
    <id>table</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>knife</id>
  <class>ph:Knife</class>
  <controller>
    <id>table</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>table</id>
  <class>ph:Table</class>
  <controller>
    <id>floor</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>chair</id>
  <class>ph:Chair</class>
  <controller>
    <id>floor</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>sofa</id>
  <class>ph:Sofa</class>
  <controller>
    <id>floor</id>
    <type>gum:Support</type>
  </controller>
</object>

<object>
  <id>floor</id>
  <class>ph:Floor</class>
</object>
```

Figure 5.15 Description des objets présents dans la scène statique

Tableau 5.5 Modélisation des objets

#	**ID**	**Class**
1	She	Human
2	spaghetti	Spaghetti
3	plate	Plate
4	soup	Soup
5	bowl	Bowl
6	glass	Glass
7	spoon	Spoon
8	fork	Fork
9	knife	Knife
10	table	Table
11	chair	Chair
12	sofa	Sofa
13	floor	Floor

Tableau 5.6 Modélisation des relations de contrôle

#	**Controller**	**Object**	**Type**
1	floor	She	Support
2	plate	spaghetti	Containment
3	table	plate	Support
4	bowl	soup	Containment
5	table	bowl	Support
6	table	glass	Support
7	table	spoon	Support
8	table	fork	Support
9	table	knife	Support
10	floor	table	Support
11	floor	chair	Support
12	floor	sofa	Support

Action

L'action principale de la scène animée est modélisée par plusieurs attributs, correspondant aux éléments qui y sont associés dans la description de scène de la section 5.4.1. Ces attributs sont d'abord les éléments de base de la description de l'action tels que *type*, *process*, *actor* et *actee*, mais aussi les informations spatiales telles que *placement*, *route*, *direction* et *motionDirection*. Puisque ces dernières doivent spécifier un ou plusieurs emplacements dans la scène virtuelle, elles sont modélisées par une paire constituée d'un objet de la scène et d'une modalité spatiale. La liste des attributs de l'action principale de notre exemple est présentée au tableau 5.7.

Tableau 5.7 Modélisation de l'action

#	**Attribut**	**Valeur**
1	type	AffectingSimpleMotion
2	process	"Ingestion"
3	actor	She
4	actee	spaghetti
5	placement	<sofa, Support>

5.4.4 Intégration de la connaissance

Une fois la scène modélisée dans notre système, la phase suivante est la formulation du problème de planification en langage PDDL. Cette phase est divisée en trois étapes expliquées dans cette section et les deux suivantes (5.4.5 et 5.4.6).

La première partie de la phase de formulation du problème consiste en la déclaration des objets de la scène jumelée à l'intégration de la connaissance sur ces derniers. Cette connaissance, obtenue à partir de l'ontologie présentée à la section 5.3.1, sert à identifier toutes les fonctionnalités offertes par les objets de la scène, représentées par les classes desquelles ces objets dérivent. Chaque classe associée à un objet de la scène sert donc à identifier un nombre de fonctionnalités définies pour celle-ci s'appliquant à cet objet. L'ensemble des fonctionnalités définies pour chaque objet correspond donc à l'ensemble de celles définies par les classes qui y sont associées. Pour chaque objet présent dans la scène, une requête est ainsi formulée à l'ontologie afin d'obtenir l'ensemble de ses classes parentes, pour ensuite les spécifier lors de la définition du problème de planification.

La définition des objets en langage PDDL consiste à spécifier l'identifiant de chaque objet et ensuite son type. Cependant, une particularité du langage nous permet de définir plusieurs types pour un seul objet. Lors de la résolution du problème, les objets peuvent donc

modifier dynamiquement leur type parmi ceux spécifiés pour correspondre aux paramètres des différentes actions du domaine.

Chacun des objets modélisés dans la scène (voir section 5.4.3) est défini pour le problème de planification. Pour notre exemple, la liste des objets définis et de leurs types est présentée au tableau 5.8

Tableau 5.8 Définition des objets et de leurs types

#	**ID**	**Types**			
1	She	Human	Object		
2	spaghetti	EatableWithForkFood	Food	Object	Spaghetti
3	plate	Container Object	Controller Plate	Dish Tableware	Flatware
4	soup	EatableWithSpoonFood	Food	Object	Soup
5	bowl	Bowl Flatware	Container Object	Controller Tableware	Dish
6	glass	Container Glassware	Controller Object	Dish Tableware	Glass
7	spoon	Dish	Object	Spoon	Utensil
8	fork	Dish	Fork	Object	Utensil
9	knife	Dish	Knife	Object	Utensil
10	table	BelowHorizontalSupport Support	Controller Table	Furniture WorkingFurniture	Object
11	chair	BelowHorizontalSupport Object	Chair SittingFurniture	Controller Support	Furniture
12	sofa	BelowHorizontalSupport SittingFurniture	Controller Sofa	Furniture Support	Object
13	floor	BelowHorizontalSupport Support	Controller	Floor	Ground

5.4.5 Détermination des prédicats initiaux

Une fois les objets de la scène définis, la seconde partie de la définition de la scène consiste à décrire l'état initial de cette dernière. Ceci est possible par la détermination des prédicats PDDL constituant l'état initial du problème. La liste des prédicats disponibles et leur signification sont expliquées à la section 5.3.2.

D'abord, le prédicat d'exécution d'action non ponctuelle doit être désactivé puisqu'aucune action n'est en cours initialement. Puis, l'acteur doit être placé à sa position de départ. Ces deux prédicats sont dits fixes puisqu'ils se retrouvent à l'initialisation de toutes les scènes. Ensuite, puisqu'aucune action n'est déjà effectuée ni en cours d'exécution au départ, les seuls prédicats se retrouvant dans l'état initial sont ceux associés aux relations de contrôle. À partir

de la modélisation de ces dernières présentée à la section 5.4.3, nous obtenons les prédicats initiaux présentés au tableau 5.9.

Tableau 5.9 Prédicats initiaux

not (**action-mutex-enabled**)
at-start She
supporting floor She
containing plate spaghetti
supporting table plate
containing bowl soup
supporting table bowl
supporting table glass
supporting table spoon
supporting table fork
supporting table knife
supporting floor table
supporting floor chair
supporting floor sofa

5.4.6 Détermination des prédicats de but

Cette partie est la dernière étape de la phase de formulation du problème de planification. Elle consiste à analyser les besoins de la scène pour identifier quels doivent être les prédicats de but du problème. Ces besoins réfèrent directement à la modélisation de l'action présentée au tableau 5.7 de la section 5.4.3.

D'abord, un prédicat d'action en cours ou d'action effectuée doit être choisi. Les prédicats sont catégorisés selon le type d'action. Puis, à partir de la valeur de l'attribut *process* de la modélisation et en concordance avec les attributs *actor* et *actee*, une action du domaine doit être choisie afin d'être effectuée dans la scène. Le choix de cette action est effectué par un algorithme s'apparentant à un système de règles permettant de déterminer quelle prédicat de but en PDDL correspond à l'action décrite ainsi qu'à ses attributs. Pour notre exemple, dans la catégorie « AffectingSimpleMotion », le *frame* « Ingestion » et l'attribut *actee* « spaghetti », de type *Spaghetti* dérivant de la classe *Food* nous indiquent qu'il s'agit de l'action de manger. Cette action est effectuée par l'acteur « She ». À défaut d'information supplémentaire, le prédicat d'action en cours sera choisi par défaut s'il y a lieu.

Ensuite, lorsque la modélisation contient de l'information supplémentaire, celle-ci doit également être représentée par un prédicat de but. Pour notre exemple, l'attribut *placement* ayant comme valeur la paire « <sofa, Support> » signifie que l'acteur « She » doit être supporté par l'objet « sofa ».

La traduction des informations des deux parties analysées ci-haut en prédicats PDDL donne les prédicats de but présentés au tableau 5.10.

Tableau 5.10 Prédicats de but

is-eating She spaghetti
supporting sofa She

5.4.7 Résolution du problème de planification

Le problème de planification étant maintenant formulé, il ne reste qu'à le résoudre pour obtenir un plan. Ce plan est le format final de transformation de l'information que nous désirons atteindre. La résolution du problème est effectuée à l'aide d'un planificateur, auquel nous fournissons la formulation du problème présentée aux sections 5.4.4, 5.4.5 et 5.4.6. Plus de détails sur le mécanisme de résolution d'un problème de planification sont fournis à la section 4.3.

Le plan obtenu après résolution du problème de planification de notre exemple est présenté au tableau 5.11.

Tableau 5.11 Plan d'animation

#	**Action**	**Paramètres**			
1	move-begin	She	floor		
2	move-to	She	fork		
3	move-end	She	floor		
4	take	She	fork	table	
5	move-begin	She	floor		
6	move-to	She	fork	plate	
7	move-end	She	floor		
8	take	She	plate	table	
9	move-begin	She	floor		
10	move-to	She	plate	sofa	
11	move-end	She	floor		
12	sit-down	She	sofa	floor	
13	eat-begin	She	spaghetti	fork	plate

CHAPITRE 6

MÉTHODOLOGIE D'ÉVALUATION

Afin de démontrer la validité de la solution proposée au chapitre 5, nous avons prévu une méthodologie d'évaluation de celle-ci. Ce chapitre a pour but de présenter cette méthodologie. Cette dernière se compose de plusieurs aspects. D'abord, nous présentons le programme développé afin de pouvoir tester notre solution. Ensuite, nous présentons les modalités de configuration des outils utilisés par le programme, c'est-à-dire la base de connaissances et le planificateur. Puis, nous expliquons l'expérience réalisée afin d'obtenir nos résultats, dont la structure et la méthode d'évaluation sont finalement présentés.

6.1 Développement du programme

Le but du programme que nous avons développé est de permettre la génération d'un plan d'animation suivant le processus présenté à la section 5.4. La figure 6.1 présente un schéma de l'architecture logicielle de notre programme.

Concrètement, ce programme traite les informations contenues dans les fichiers d'entrée XML (*Scene Declaration* et *Scene Objects*) afin de formuler un problème de planification, contenu dans deux fichiers PDDL. Ces fichiers sont ensuite fournis à un planificateur afin de permettre la résolution du problème. Finalement, le programme reprend le résultat de la planification afin de produire un fichier texte représentant le plan d'animation.

Nous décrivons ici les différents éléments de l'architecture logicielle.

Fichiers d'entrée XML Ces deux fichiers correspondent à la description de la scène à traiter. Pour chaque scène à animer, une description conceptuelle abstraite ainsi qu'une liste d'objets de la scène doivent être fournis. Ces dernières correspondent aux formats de représentation présentés dans notre solution proposée aux sections 5.1 et 5.2.

PlanGenerator Ce module constitue l'essentiel de notre programme. Il s'agit du logiciel, conçu en Java, effectuant tout le traitement nécessaire pour la génération du plan d'animation, à l'exception de la résolution du problème de planification. Dans le schéma, nous avons présenté les différents paquetages composant notre programme. D'abord, le paquetage *reader* permet la lecture des fichiers d'entrée XML et leur traitement. Ensuite, les paquetages *declaration* et *action* définissent les classes permettant la modélisation de la scène expliquée à la section 5.4.3. La liste de ces classes est la suivante :

– AffectingSimpleMotion

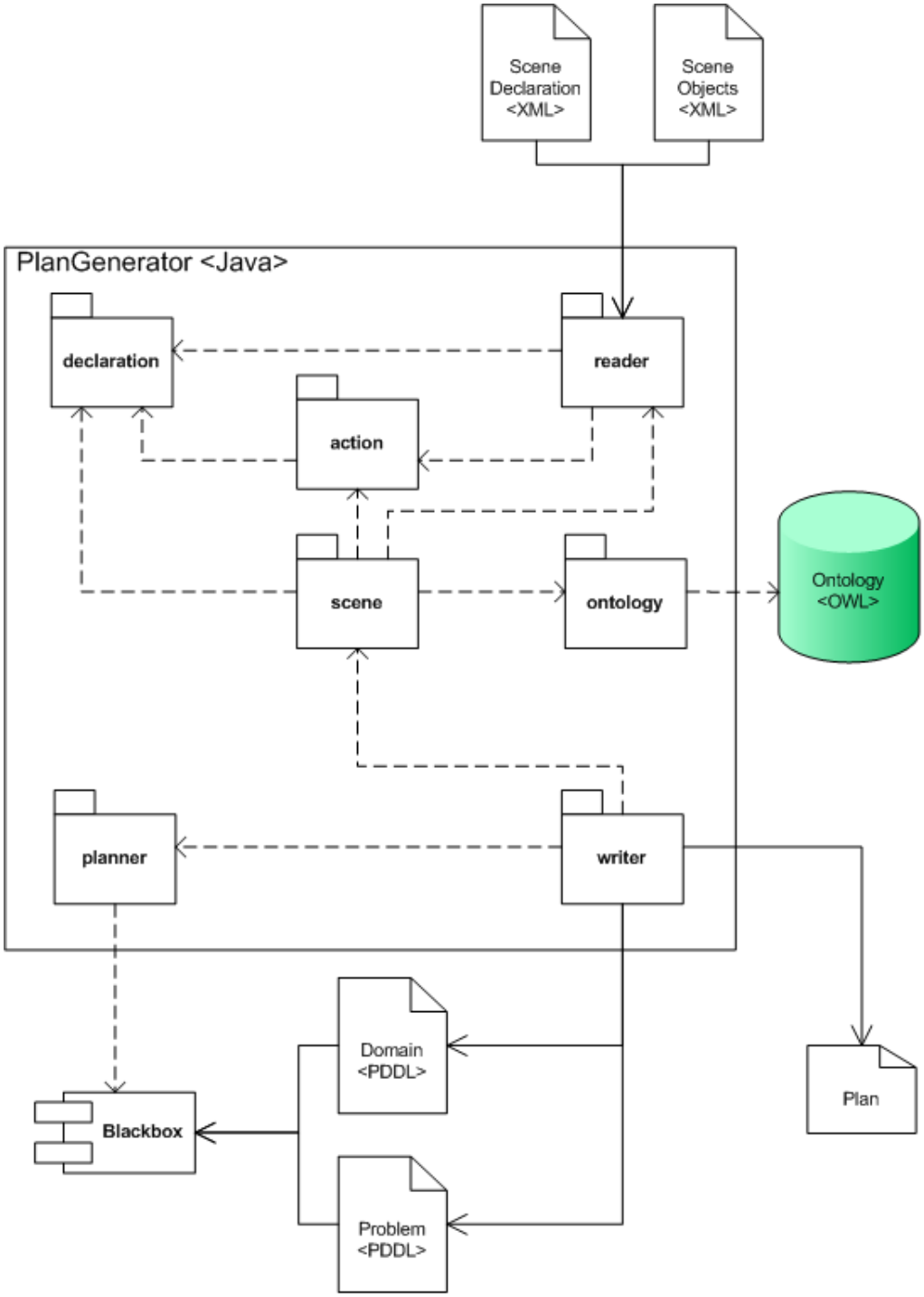

Figure 6.1 Architecture logicielle du programme

- AffectingDirectedMotion
- AffectingOrientationChange
- NonAffectingSimpleMotion
- NonAffectingDirectedMotion
- NonAffectingOrientationChange
- ControlRelation
- ObjectDeclaration
- LocationDeclaration
- RouteDeclaration
- SpatialModalityDeclaration

Le paquetage *scene* est le paquetage central de notre programme, c'est-à-dire celui contenant la modélisation concrète de la scène. La classe statique *Scene* contient toutes les informations de la scène et permet donc de répondre aux requêtes des classes du paquetage *writer*, contenant à la fois les mécanismes de formulation du problème et d'écriture du plan. Le paquetage *scene* fait aussi appel au paquetage *ontology* permettant l'interaction avec la base de connaissances. Finalement, le paquetage *planner* encapsule l'utilisation du planificateur et la résolution du problème.

Ontologie La base de connaissances de type ontologie est indépendante de notre programme et implémentée en langage OWL. Sa structure correspond à la représentation de la connaissance présentée à la section 5.3.1 de notre solution proposée. Les détails d'interaction avec l'ontologie sont expliqués à la section 6.2.

Fichiers PDDL Ces deux fichiers correspondent à la formulation du problème de planification, donc l'étape finale du processus de traitement avant la résolution du problème. Le fichier de domaine correspond au domaine d'actions présenté à la section 5.3.2 de notre solution proposée et est indépendant de la scène à animer. De son côté, le fichier de problème représente directement la scène à animer sous forme de problème de planification à résoudre. Il s'agit de la formulation en PDDL des informations présentées dans les sections 5.4.4, 5.4.5 et 5.4.6.

Planificateur L'outil utilisé pour la résolution du problème de planification est le planificateur Blackbox (Kautz et Selman (1998)). Ce dernier nécessite en entrée les deux fichiers PDDL de domaine et de problème mentionnés ci-haut pour en traiter le contenu et ainsi fournir un plan. Les détails de configuration et d'utilisation du planificateur sont donnés à la section 6.3.

Plan Le dernier fichier produit par notre programme est le plan, phase finale de l'information dans le pipeline de notre projet. Ce plan constitue simplement le résultat formaté du planificateur. Le format et la structure de ces résultats sont présentés à la section 6.5.

6.2 Utilisation de la base de connaissances

La base de connaissances de type ontologie que nous avons conçue est une composante distincte de notre programme dans notre architecture logicielle puisqu'elle est complètement indépendante de ce dernier. Elle est aussi implémentée en langage OWL, conçu sur mesure pour la création d'ontologies. Des mécanismes d'interaction doivent donc être prévus entre le programme et la base de connaissances.

Cette section présente les paramètres de configuration nécessaires à l'interaction avec l'ontologie par notre programme, ainsi que des informations supplémentaires sur les requêtes formulées pour l'exécution des scénarios de test (voir section 6.4).

6.2.1 Configuration

Afin d'interagir avec une ontologie en langage OWL à partir d'un programme en Java, il est nécessaire d'utiliser une interface de programmation. Celle que nous utilisons se nomme OWL API et a été conçue par Horridge et Bechhofer (2009). Cette interface de programmation (API) permet la création et la manipulation d'ontologies en langage OWL à partir d'un programme en Java. Dans notre cas, uniquement l'aspect de formulation de requêtes à l'ontologie est utilisé, puisque nous ne cherchons pas à créer ou modifier du contenu dans notre base de connaissances au cours de l'exécution du programme.

L'interface OWL API nous permet de charger une ontologie à l'exécution du programme à partir d'un fichier local ou d'un emplacement spécifié (IRI). Elle charge ensuite les données en mémoire et les indexe sous forme de base de données interne afin d'accélérer les requêtes et les inférences.

La deuxième partie de la configuration consiste en l'utilisation d'un raisonneur sémantique. Celui-ci permet d'effectuer des inférences sur l'ensemble de données que constitue une ontologie. Ces inférences peuvent concerner une hiérarchie de classes, de propriétés ou d'attributs, ainsi que les caractéristiques des individus s'il y a lieu. Dans notre cas, seules les inférences de type hiérarchie de classes nous intéressent.

La raisonneur sémantique que nous utilisons se nomme HermiT, conçu par Shearer *et al.* (2008). Ce dernier possède également une interface de programmation en Java et est compatible avec OWL API. L'utilisation de ces deux interfaces nous permet donc de formuler des requêtes à l'ontologie pendant l'exécution de notre programme.

6.2.2 Requêtes formulées

Une fois le raisonneur sémantique bien configuré, nous pouvons utiliser l'interface de programmation pour formuler des requêtes à l'ontologie. L'interface OWLReasoner de OWL

API encapsule différentes requêtes possibles et est implémentée par notre raisonneur HermiT.

Un exemple de requête effectuée au cours de l'exécution de notre programme consiste à obtenir toutes les classes présentes dans l'ontologie, pour la définition des types du domaine. Pour ce faire, les appels de fonctions *getTopClassNode* et *getSubClasses* de l'interface OWLReasoner sont utiles. En spécifiant que nous voulons obtenir toutes les sous-classes et non uniquement les sous-classes directes, nous obtenons les informations souhaitées.

Un second exemple de requête, la plus souvent utilisée, est celle permettant d'obtenir les classes parentes d'une classe particulière. Comme mentionné à la section 5.3.1, la hiérarchie de classes des objets de l'ontologie est axée sur la fonctionnalité de ceux-ci, c'est-à-dire qu'un objet possède comme fonctionnalités celles de l'ensemble de ses classes parentes. Alors, comme expliqué à la section 5.4.4, nous avons besoin d'obtenir tous les types correspondant à chaque objet de la scène lors de la formulation du problème de planification. Cette requête est effectuée grâce aux appels de fonctions *getOWLClass* de l'interface OWLDataFactory et *getSuperClasses* de l'interface OWLReasoner. En spécifiant que nous voulons obtenir toutes les classes parentes et non uniquement les classes parentes directes, nous obtenons les informations souhaitées.

D'autres mécanismes permettent d'effectuer des requêtes plus complexes à l'ontologie, par exemple de formuler des requêtes en langage SPARQL (voir section 3.1.2), mais le niveau de complexité de nos requêtes ne requerrait pas d'utiliser de tels mécanismes. Nous nous en sommes donc tenus à OWL API, ce dernier étant très convivial pour les requêtes simples.

6.3 Configuration du planificateur

La résolution du problème de planification, présentée à la section 5.4.7 de notre solution proposée, a été effectuée avec le planificateur Blackbox (Kautz et Selman (1998)). Ce dernier est disponible au public et s'exécute en ligne de commande grâce à la commande suivante, où les deux fichiers PDDL sont ceux présents dans le schéma d'architecture logicielle (voir section 6.1) :

```
blackbox.exe -o domain.pddl -f problem.pddl
```

Afin d'exécuter cette commande et d'en retirer les résultats à partir de notre programme en langage Java, nous utilisons les classes Process et Runtine de *java.lang*, pour l'exécution, ainsi que BufferedReader et InputStreamReader de *java.io*, pour obtenir les données de sortie.

Les paramètres du problème de planification sont définis dans le fichier PDDL de domaine. Dans notre cas, nous utilisons les paramètres *strips* et *typing*. Le premier signifie que nous utilisons un formalisme STRIPS pour la définition des prédicats et des actions, tandis que

le second indique l'utilisation du typage dans les déclarations des objets et des paramètres d'actions. Plus de détails sur ce formalisme sont donnés à la section 4.2 de notre revue de littérature.

6.4 Expérience réalisée

Les trois sections précédentes expliquent en détails techniques l'environnement utilisé pour l'implémentation de notre solution. L'étape suivante de notre méthodologie d'évaluation consiste à réaliser une expérience afin d'obtenir des résultats concrets. Pour ce faire, nous avons identifié 16 situations précises à traiter par notre programme afin d'obtenir un plan d'animation.

6.4.1 Contraintes posées

Pour démontrer la faisabilité de notre solution, les situations que nous avons identifiées devaient respecter certaines contraintes posées par notre implémentation. D'abord, puisque la base de connaissances et le domaine d'actions ne couvrent pas exhaustivement le monde réel, le choix des situations devait se restreindre à l'utilisation d'actions répertoriées dans le domaine et d'objets présents dans l'ontologie. Ensuite, chaque situation devait pouvoir être exprimée dans notre format de représentation conceptuelle abstraite (voir section 5.1), dont les spécifications sont présentées à l'annexe A.

En plus de ces contraintes de base posées par les limitations de notre domaine d'application, quelques contraintes supplémentaires sont posées par des choix que nous avons effectués. D'abord, parmi les huit types de configurations permises par notre schéma de validation, nous nous sommes concentrés sur les six types impliquant une action de type mouvement. Les scénarios de test n'impliquent donc pas de configurations de type *SpatialLocating* ni *Spatial-TemporalLocating*, ou aucune action concrète n'est représentée. De plus, dans le but d'axer nos scénarios de test sur les différentes actions possibles et *frames* disponibles plutôt que sur les attributs, nous avons choisi de limiter les attributs implémentés à *placement*, *route*, *source*, *destination*, *motionDirection* et *orientationDirection* et de laisser de côté les attributs *direction*, *orientationRoute* et *pathPlacement*. Cette décision se justifie par le fait que ces attributs complexifiaient l'implémentation des cas de tests et ce, sans apport supplémentaire tangible à la preuve de faisabilité de notre solution, puisque les six attributs conservés couvrent déjà la grande majorité des cas.

Finalement, nous avons posé des contraintes sur les valeurs possibles des types de contrôle et modalités spatiales pouvant être spécifiées dans les scénarios de test. D'abord, nous avons restreint les types de contrôle aux deux types que nous avons définis dans notre base de

connaissances : *Support* et *Containment*. Ensuite, les modalités spatiales permises incluent bien sûr les deux types de contrôle, mais aussi les types *Proximal* et *GeneralDirectional*, indiquant respectivement une proximité et une direction.

6.4.2 Scénarios de test

La structure des scénarios de test consiste en la définition d'une description de scène et d'une liste d'objets de la scène statique pour chacune des situations à traiter. Afin de représenter les situations que nous avons choisies pour les scénarios, nous présentons chacune d'entre elles sous forme de modélisation de l'action de la scène (voir section 5.4.3) et montrons ainsi les informations pertinentes de leur description. Nous supposons que pour chaque description de scène modélisée, une liste d'objets de la scène statique a été fournie pour permettre la modélisation de la scène et les étapes subséquentes de la génération du plan d'animation.

La liste des scénarios de test est présentée aux tableaux 6.1 à 6.16. Les scénarios ont été sélectionnés dans le but de couvrir une grande partie des possibilités offertes par notre ontologie et notre domaine d'actions. Ainsi, les situations varient selon les paramètres suivants :

- Type de configuration
- Action principale (*frame*)
- Attributs de l'action
- Objets impliqués
- Scène statique fournie

Tableau 6.1 Scénario de test 1 - « He is dancing »

#	**Attribut**	**Valeur (- Classe)**
1	type	NonAffectingSimpleMotion
2	process	”Self-motion”
3	actor	He - Human

Tableau 6.2 Scénario de test 2 - « He is dancing on the ground »

#	**Attribut**	**Valeur (- Classe)**
1	type	NonAffectingSimpleMotion
2	process	”Self-motion”
3	actor	He - Human
4	placement	<ground - Ground, Support>

Tableau 6.3 Scénario de test 3 - « He pushed his washer next to his dryer »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingDirectedMotion
2	process	”Cause-motion”
3	actor	He - Human
4	actee	washer - Washer
5	route-destination	<dryer - Dryer, Proximal>

Tableau 6.4 Scénario de test 4 - « He puts the ball in the box »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingDirectedMotion
2	process	”Placing”
3	actor	He - Human
4	actee	ball - Ball
5	route-destination	<box - Box, Containment>

Tableau 6.5 Scénario de test 5 - « He stands up »

#	**Attribut**	**Valeur (- Classe)**
1	type	NonAffectingDirectedMotion
2	process	”Change-posture”
3	actor	He - Human

Tableau 6.6 Scénario de test 6 - « John pointed the camera at Lucy »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingOrientationChange
2	process	”Aiming”
3	actor	John - Human
4	actee	camera - Camera
5	orientationDirection	<Lucy - Human, GeneralDirectional>

Tableau 6.7 Scénario de test 7 - « Mark eats an apple »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingSimpleMotion
2	process	”Ingestion”
3	actor	Mark - Human
4	actee	apple - Apple

Tableau 6.8 Scénario de test 8 - « Mary takes a shower »

#	**Attribut**	**Valeur (- Classe)**
1	type	NonAffectingSimpleMotion
2	process	”Grooming”
3	actor	Mary - Human
4	placement	<shower - Shower, Containment>

Tableau 6.9 Scénario de test 9 - « Peter took a book from the shelf to bed »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingDirectedMotion
2	process	”Bringing”
3	actor	Peter - Human
4	actee	book - Book
5	route-source	<shelf - Shelf, Support>
6	route-destination	<bed - Bed, Support>

Tableau 6.10 Scénario de test 10 - « She ate her spaghetti on the sofa »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingSimpleMotion
2	process	”Ingestion”
3	actor	She - Human
4	actee	spaghetti - Spaghetti
5	placement	<sofa - Sofa, Support>

Tableau 6.11 Scénario de test 11 - « She moved from the refrigerator to the table »

#	**Attribut**	**Valeur (- Classe)**
1	type	NonAffectingDirectedMotion
2	process	”Motion”
3	actor	She - Human
4	route-source	<refrigerator - Refrigerator, Proximal>
5	route-destination	<table - Table, Proximal>

Tableau 6.12 Scénario de test 12 - « She throws the ball at the wall »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingDirectedMotion
2	process	”Cause-motion”
3	actor	She - Human
4	actee	ball - Ball
5	motionDirection	<wall - Wall, GeneralDirectional>

Tableau 6.13 Scénario de test 13 - « The boy sits down on the sofa »

#	**Attribut**	**Valeur (- Classe)**
1	type	NonAffectingDirectedMotion
2	process	”Change-posture”
3	actor	boy - Human
4	route-destination	<sofa - Sofa, Support>

Tableau 6.14 Scénario de test 14 - « The girl lied down on her bed »

#	**Attribut**	**Valeur (- Classe)**
1	type	NonAffectingDirectedMotion
2	process	”Change-posture”
3	actor	girl - Human
4	route-destination	<bed - Bed, Support>

Tableau 6.15 Scénario de test 15 - « Vicky drinks water »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingSimpleMotion
2	process	”Ingestion”
3	actor	Vicky - Human
4	actee	water - Water

Tableau 6.16 Scénario de test 16 - « William washes the dishes »

#	**Attribut**	**Valeur (- Classe)**
1	type	AffectingSimpleMotion
2	process	”Grooming”
3	actor	William - Human
4	actee	dishes - Dish

6.5 Structure des résultats

Grâce au programme que nous avons implémenté, dont l'architecture logicielle est présentée à la section 6.1, nous pouvons obtenir des résultats à partir de scénarios respectant les balises de notre expérience expliquées à la section 6.4.1. Après avoir choisi les scénarios de test de la section 6.4.2, créé les descriptions de scènes correspondantes et généré les listes d'objets de la scène statique appropriées, il nous est possible d'obtenir les plans d'animations pour chacun des scénarios en fournissant ces deux fichiers à notre programme. Nous présentons ici la structure de ces résultats.

Les plans d'animations générés par notre programme contiennent la liste des actions à effectuer par l'acteur afin d'obtenir une animation illustrant la situation décrite initialement. Leur structure consiste donc en une liste ordonnée d'actions du domaine suivies de leurs paramètres, représentant les éléments de la scène concernés par l'action. La spécification des actions, des paramètres et de leurs types respectifs est donnée dans la définition du domaine d'actions, à l'annexe C.

Un exemple de résultat a été fourni lors de l'explication du processus de résolution du problème de planification, à la section 5.4.7 de notre solution proposée. La structure exacte des résultats n'est cependant pas représentée dans un tableau, mais fourni sous forme de liste. Le plan généré pour notre exemple de la section 5.4 a donc en réalité la structure suivante :

```
Begin plan
1 (move-begin she floor)
2 (move-to she fork)
3 (move-end she floor)
4 (take she fork table)
5 (move-begin she floor)
6 (move-to she fork plate)
7 (move-end she floor)
8 (take she plate table)
9 (move-begin she floor)
10 (move-to she plate sofa)
11 (move-end she floor)
12 (sit-down she sofa floor)
13 (eat-begin she spaghetti fork plate)
End plan
```

Tous les plans générés à partir de notre programme ont une structure identique à celle-ci. Une fois ces derniers obtenus, nous pouvons procéder à une évaluation des résultats selon les balises proposées à la section 6.6.

6.6 Évaluation des résultats

Une fois les résultats des scénarios de test de la section 6.4.2 obtenus, une évaluation de ces derniers doit être effectuée avec de remplir les objectifs O6 et O7 de la problématique, énoncés à la section 2.5. D'abord, nous proposons une validation des plans afin de vérifier la faisabilité de la génération d'un plan d'animation à partir d'une représentation conceptuelle abstraite. Ensuite, nous proposons d'analyser la couverture de la ressource *FrameNet* par notre solution afin de démontrer l'applicabilité de notre solution à une partie suffisante du domaine de conceptualisation des actions.

6.6.1 Validation des plans

Les plans générés grâce à notre solution consistent en des listes d'actions à effectuer afin de représenter virtuellement la situation décrite au départ par une représentation conceptuelle abstraite. Afin d'effectuer la validation de ces plans, nous devons évaluer si l'animation que chacun d'entre eux engendre respecte la description de scène initiale. Pour ce faire, nous proposons une validation manuelle de chacun des scénarios de test. L'échantillon de 16 situations initiales créé pour les scénarios de test est suffisamment petit pour nous permettre de valider manuellement chaque plan généré et de vérifier que les actions incluses représentent adéquatement chacune des situations.

Il est nécessaire d'examiner la liste d'actions afin de déterminer que la situation à représenter est bien couverte par celle-ci. Comme la compréhension d'une situation exprimée dans une scène virtuelle relève de l'esprit d'analyse humain, nous sommes en mesure d'évaluer si une scène exprime correctement une situation donnée. Le but du projet n'est pas de déterminer le niveau de compréhension optimal, mais plutôt de générer une scène animée répondant aux critères de la description conceptuelle. Cette vérification vise donc à déterminer qu'aucune erreur ne s'est glissée dans le plan d'animation.

Les plans obtenus grâce au planificateur Blackbox sont garantis d'être valides sur le plan de l'exécution, sans quoi celui-ci aurait indiqué une impossibilité de résoudre le problème et n'aurait fourni aucun plan. Nous pouvons donc affirmer que la génération réussie d'un plan par le planificateur assure la résolubilité du problème de planification formulé. La validation des plans sert donc ultimement à vérifier que le problème de planification formulé est correct.

Une fois les plans évaluées manuellement, nous serons en mesure d'affirmer si l'objectif O6 de notre problématique a été atteint ou non. La formulation exacte de cet objectif consiste à « vérifier la faisabilité de la génération d'un plan d'animation à partir d'une représentation conceptuelle abstraite ».

6.6.2 Couverture du domaine d'actions

Cet aspect de notre méthodologie d'évaluation consiste à parcourir la ressource *FrameNet* (Baker *et al.* (1998)), dont les *frames* servent à indiquer l'action de la scène dans notre solution proposée (voir section 5.1), afin de déterminer quelle proportion de l'ensemble de ces *frames* est traitable par notre solution. Ceci sert à vérifier l'atteinte de l'objectif O7 de notre problématique, dont la formulation exacte est « démontrer l'applicabilité de notre solution à une partie suffisante du domaine de conceptualisation des actions ».

Le point de départ que nous avons choisi pour notre parcours de *FrameNet* est *Event*, le *frame* de base dans la hiérarchie de *FrameNet* pour représenter les actions. Nous avons ensuite parcouru chaque *frame* descendant de *Event* afin de le classifier selon plusieurs catégories. Ces dernières représentent les types d'actions de *GUM* utilisés dans notre solution (voir section 5.1.2) pour représenter les actions de mouvement. Chaque *frame* est donc associé à une catégorie d'actions de *GUM* ou, à défaut de pouvoir être catégorisé selon cette classification, placé dans une des catégories à part représentant d'autres types d'actions ne pouvant pas être traitées par notre système.

La méthode d'évaluation consiste ensuite à identifier, parmi tous les *frames* classifiés, d'abord lesquels sont pris en charge par notre implémentation, mais aussi, puisque notre solution n'est qu'une preuve de concept, lesquels sont potentiellement traitables selon notre évaluation. Nous sommes ensuite en mesure de fournir un calcul de la proportion des *frames* couverts par notre solution. Ce calcul peut aussi tenir compte de ceux que nous ne désirons pas traiter à l'aide d'un tel système. Plus d'explications sont données à la section 7.8.

La classification des *frames* est la suivante :

NonAffectingSimpleMotion *Grooming, Self-motion.*

NonAffectingDirectedMotion *Arriving, Change-posture, Cotheme, Departing, Motion, Motion-directional, Self-motion.*

NonAffectingOrientationChange *Change-direction, Moving-in-place.*

AffectingSimpleMotion *Attack, Cause-to-be-sharp, Cause-to-fragment, Closure, Create-physical-artwork, Cutting, Destroying, Detonate-explosive, Grinding, Grooming, Hostile-encounter, Ingest-substance, Ingestion, Operating-a-system, Setting-fire, Text-creation, Using.*

AffectingDirectedMotion *Apply-heat, Attaching, Bringing, Burying, Cause-impact, Cause-motion, Cause-to-be-dry, Cause-to-be-wet, Change-tool, Cooking-creation, Create-representation, Downing, Emptying, Exchange, Filling, Getting, Giving, Hiding-objects, Hit-target, Immobilization, Mass-motion, Operate-vehicule, Passing, Placing, Removing, Replacing, Soaking, Shoot-projectiles, Taking, Transfer, Use-firearm.*

AffectingOrientationChange *Aiming, Cause-to-move-in-place.*

Action sensorielle *Cause-bodily-experience, Cause-to-experience, Cause-to-make-noise, Cause-to-perceive, Cause-to-wake, Communication, Experience-bodily-harm, Heralding, Passing-off, Perception-active, Response, Silencing, Waking-up.*

Action morale *Arrested, Assistance, Atonement, Bail-decision, Choosing, Clemency, Collaboration, Coming-to-believe, Confronting-problem, Daring, Education-teaching, Endangering, Execution, Extradition, Forging, Front-for, Imposing-obligation, Imprisonment, Inhibit-movement, Intercepting, Killing, Manipulate-into-doing, Misdeed, Name-conferral, Objective-influence, Releasing, Reparation, Rewards-and-punishment, Ruling-legally, Visiting.*

Action d'état *Achieving-first, Adjusting, Aging, Becoming, Becoming-a-member, Causation, Cause-change, Cause-change-of-consistency, Cause-change-of-phase, Cause-change-of-position-on-a-scale, Cause-change-of-strength, Cause-expansion, Cause-temperature-change, Cause-to-be-included, Cause-to-rot, Change-of-consistency, Corroding-caused, Cure, Damaging, Duplication, Exhaust-resource, Firing, Forming-relationships, Get-a-job, Go-into-shape, Hiring, Historic-event, Manipulate-into-shape, Practice, Processing-materials, Quiting, Rejuvenation, Render-nonfuntional, Rotting, Translating, Working-a-post.*

Action de l'espace *Arranging, Avoiding, Cause-to-amalgamate, Change-accessibility, Dispersal, Examination, Fleeing, Gathering-up, Traversing.*

Action complexe *Birth, Building, Cause-to-continue, Cause-to-end, Cause-to-resume, Death, Execute-plan, Interrupt-process, Manufacturing, Military-operation, Process-end, Process-pause, Process-resume, Process-start, Process-stop, Resolve-problem.*

CHAPITRE 7

RÉSULTATS ET DISCUSSION

À la suite de la réalisation de l'expérience expliquée à la section 6.4, nous avons obtenu des résultats consistant en des plans d'animation pour les scénarios de test générés. Dans un premier temps, ce chapitre a pour but de présenter ces résultats et ensuite d'en analyser le contenu. Puis, dans un second temps, ce chapitre propose un discussion axée sur les principaux aspects de notre projet, de la solution proposée à la méthodologie d'évaluation.

D'abord, nous présentons les plans obtenus à partir du traitement des scénarios de test présentés à la section 6.4.2, pour ensuite en faire l'analyse. Nous poursuivons avec une discussion en quatre portions sur l'impact des différents aspects de la solution proposée : les représentations abstraites, les ressources sémantiques, la représentation des connaissances et la planification classique. Puis, nous discutons des limitations de la méthodologie d'évaluation présentée au chapitre 6, pour ensuite analyser la couverture du domaine d'actions présentée à la section 6.6.2.

Finalement, nous proposons un bref retour sur nos objectifs de recherche afin de déterminer si ces derniers ont été atteints grâce à notre solution proposée et notre méthodologie d'évaluation.

7.1 Plans générés

Voici la liste des plan générés par notre programme à la suite du traitement des 16 scénarios de test présentés à la section 6.4.2. La structure de chaque plan est conforme à celle expliquée à la section 6.5.

Plan du scénario de test 1 - « He is dancing »

```
Begin plan
1 (move-begin he floor)
End plan
```

Plan du scénario de test 2 - « He is dancing on the ground »

```
Begin plan
1 (move-begin he ground)
End plan
```

Plan du scénario de test 3 - « He pushed his washer next to his dryer »

```
Begin plan
1 (move-begin he floor)
2 (move-to he washer)
3 (move-end he floor)
4 (push-to he washer dryer)
End plan
```

Plan du scénario de test 4 - « He puts the ball in the box »

```
Begin plan
1 (move-begin he floor)
2 (move-to he ball)
3 (move-end he floor)
4 (take he ball table)
5 (move-begin he floor)
6 (move-to he ball box)
7 (move-end he floor)
8 (place-in he ball box)
End plan
```

Plan du scénario de test 5 - « He stands up »

```
Begin plan
1 (stand-up he sofa floor)
End plan
```

Plan du scénario de test 6 - « John pointed the camera at Lucy »

```
Begin plan
1 (move-begin john floor)
2 (move-to john camera)
3 (move-end john floor)
4 (take john camera table)
5 (aim-begin john camera lucy)
End plan
```

Plan du scénario de test 7 - « Mark eats an apple »

```
Begin plan
1 (move-begin mark floor)
2 (move-to mark apple)
3 (move-end mark floor)
4 (take mark apple table)
5 (eat-begin mark apple)
6 (eat-end mark apple)
End plan
```

Plan du scénario de test 8 - « Mary takes a shower »

```
Begin plan
1 (move-begin mary floor)
2 (move-to mary shower)
3 (move-end mary floor)
4 (get-in mary shower floor)
5 (wash-itself-begin mary shower)
End plan
```

Plan du scénario de test 9 - « Peter took a book from the shelf to bed »

```
Begin plan
1 (move-begin peter floor)
2 (move-to peter book)
3 (move-end peter floor)
4 (take peter book shelf)
5 (move-begin peter floor)
6 (move-to peter book bed)
7 (move-end peter floor)
8 (lie-down peter bed floor)
End plan
```

Plan du scénario de test 10 - « She ate her spaghetti on the sofa »

```
Begin plan
1 (move-begin she floor)
2 (move-to she fork)
3 (move-end she floor)
4 (take she fork table)
5 (move-begin she floor)
6 (move-to she fork plate)
7 (move-end she floor)
8 (take she plate table)
9 (move-begin she floor)
10 (move-to she plate sofa)
11 (move-end she floor)
12 (sit-down she sofa floor)
13 (eat-begin she spaghetti fork plate)
End plan
```

Plan du scénario de test 11 - « She moved from the refrigerator to the table »

```
Begin plan
1 (move-begin she floor)
```

```
2 (move-to she refrigerator)
3 (move-to she refrigerator table)
End plan
```

Plan du scénario de test 12 - « She throws the ball at the wall »

```
Begin plan
1 (move-begin she floor)
2 (move-to she ball)
3 (move-end she floor)
4 (take she ball box)
5 (aim she ball wall)
6 (throw-at she ball wall)
End plan
```

Plan du scénario de test 13 - « The boy sits down on the sofa »

```
Begin plan
1 (move-begin boy floor)
2 (move-to boy sofa)
3 (move-end boy floor)
4 (sit-down boy sofa floor)
End plan
```

Plan du scénario de test 14 - « The girl lied down on her bed »

```
Begin plan
1 (move-begin girl floor)
2 (move-to girl bed)
3 (move-end girl floor)
4 (lie-down girl bed floor)
End plan
```

Plan du scénario de test 15 - « Vicky drinks water »

```
Begin plan
1 (move-begin vicky floor)
2 (move-to vicky glass)
3 (move-end vicky floor)
4 (take vicky glass countertop)
5 (move-begin vicky floor)
6 (move-to vicky glass tap)
7 (move-end vicky floor)
8 (pour vicky water tap glass)
9 (drink-begin vicky water glass)
End plan
```

Plan du scénario de test 16 - « William washes the dishes »

```
Begin plan
1 (move-begin william floor)
2 (move-to william dishes)
3 (move-end william floor)
4 (take william dishes countertop)
5 (move-begin william floor)
6 (move-to william dishes sink)
7 (move-end william floor)
8 (wash-begin william dishes sink)
End plan
```

7.2 Analyse des résultats

La première constatation que nous pouvons effectuer à la suite de l'examen des résultats de la section 7.1 est que pour chacun des scénarios de test définis à la section 6.4.2, il a été possible de générer un plan d'animation complet. Comme mentionné à la section 6.6.1, cela signifie qu'il a été possible, pour chacun des scénarios, de formuler un problème de planification résoluble par le planificateur. Cependant, avant d'affirmer que chacune des situations exprimées dans les scénarios de test est compatible avec notre solution, nous devons vérifier la validité des plans fournis au niveau du réalisme dans le monde réel. Cette démarche permet de vérifier que les plans générés, bien que complets et conformes au domaine spécifié, ne comportent pas d'incongruités quant à leur possible déroulement dans le monde réel. Il s'agit donc également d'une validation du domaine que nous avons défini afin qu'il ne comporte pas d'erreurs dans la modélisation du monde, que ce soit sur le plan des objets ou des actions de la scène. L'analyse qui suit est donc basée sur une interprétation des séquences d'actions contenues dans les plans afin d'évaluer leur plausibilité et leur réalisme.

Nous évaluerons les plans selon les catégories des actions représentées dans les situations initiales de leurs scénarios respectifs. Nous discuterons ensuite du respect des contraintes posées par ces situations initiales dans les plans générés.

NonAffectingSimpleMotion

Les plans 1, 2 et 8 représentent des actions de type NonAffectingSimpleMotion. Les problèmes de planification engendrés par celles-ci sont les plus simples à résoudre puisqu'elles n'impliquent ni élément affecté (*actee*), ni route ou direction précise à intégrer dans l'animation. D'ailleurs, les résultats obtenus pour les scénarios de test 1 et 2 sont des plans contenant une seule action. En effet, la seule action à représenter dans ces scénarios est l'acteur qui danse.

Le scénario 2 spécifie un emplacement, mais l'acteur s'y retrouve déjà initialement alors une seule action est nécessaire. Ces plans sont cohérents avec les situations initiales puisque ces dernières ne posent qu'une seule contrainte, c'est-à-dire le mouvement de l'acteur, sans distinction à son type. Cette absence de distinction entre les types de mouvements est discutée à la section 7.4.2, mais les plans 1 et 2 restent valides.

L'autre scénario dans cette catégorie est le scénario 8, impliquant l'action de se laver ayant comme emplacement une douche, nécessite quelques actions supplémentaires. Ces actions consistent à se rendre à la douche, y entrer et à s'y laver. Ceci est également cohérent avec les contraintes initiales d'action et d'emplacement spécifiées dans la description de scène. Nous pouvons donc affirmer que les actions de type NonAffectingSimpleMotion testées sont correctement représentées dans les plans d'animations générées par notre solution.

NonAffectingDirectedMotion

Des actions de type NonAffectingDirectedMotion sont représentées dans les scénarios 5, 11, 13 et 14. Celles-ci n'impliquent pas non plus d'élément affecté, mais le déroulement spatial de l'action est plus complexe dû à la spécification d'une route.

D'abord, le scénario 5 représente l'action de se lever. Cette action est catégorisée *DirectedMotion* plutôt que *SimpleMotion* car il est possible de spécifier le point initial de l'action, qui est modélisé comme l'attribut *source* de la route. Cependant, dans le cas de ce scénario, aucun attribut route n'est spécifié, ce qui réduit les contraintes posées à la seule action de se lever. En ayant l'acteur assis dans l'état initial de la scène, la seule action nécessaire est de se lever, ce qui résulte en un plan à une seule action.

Étudions maintenant le cas du scénario 11 comportant une route complète, c'est-à-dire un attribut *source* et *destination*. Un tel cas pose plusieurs contraintes, dont l'action à effectuer, mais aussi que cette dernière s'effectue tout au long de la route spécifiée. La description de scène indique que l'action de mouvement doit être effectuée à partir du réfrigérateur jusqu'à la table. Le plan généré à partir de cette description contient 3 actions, dont les 2 actions spécifiant que l'acteur se présente au réfrigérateur et ensuite à la table, toujours en mouvement. Les contraintes sont donc respectées.

Enfin, les scénarios 13 et 14 représentent des situations similaires où l'acteur doit effectuer une action correspondant au *frame* « Change-posture ». Le défi de ces deux scénarios de test réside dans le fait que le système doit pouvoir détecter de quelle action spécifique il s'agit entre les possibilités *stand-up*, *sit-down* et *lie-down* découlant de ce *frame*. Dans les deux cas, un attribut *destination* est spécifié avec une modalité spatiale de support, mais pointant vers un objet de type différent. Le scénario 13 implique un sofa, ce qui signifie que l'acteur doit s'y asseoir, tandis que le scénarios 14 indique un lit où l'acteur doit s'y coucher. Les deux

plans générés incluent le déplacement de l'acteur vers l'objet concerné, puis l'action spécifique au type de support offert, c'est-à-dire *sit-down* pour le plan 13 et *lie-down* pour le plan 14. Malgré l'ambiguïté de l'action à effectuer dans la description de scène, les plans générés sont valides et cohérents avec les objets de la scène. La sous-spécification des *frames* causant cette ambiguïté est discutée à la section 7.4.2.

AffectingSimpleMotion

Des actions de type AffectingSimpleMotion sont représentées dans les scénarios 7, 10, 15 et 16. Celles-ci impliquent un élément affecté dans la scène, mais ne spécifient pas de route. Les seules contraintes possibles sont donc l'obtention ou l'atteinte de l'élément affecté, l'emplacement spatial et les préconditions à l'exécution des actions.

D'abord, les scénarios 7 et 10 impliquent l'action de manger. Dans les deux cas, les plans comportent donc les actions nécessaires à l'obtention de la nourriture concernée. Le défi des cas comportant l'action de manger est l'utilisation des couverts et ustensiles correctement. Ainsi, l'action de manger une pomme contenue dans le plan 7 comporte comme précondition le fait de la tenir dans sa main, tandis que l'actions de manger du spaghetti du plan 10 implique de tenir une fourchette tout en ayant obtenu ou atteint l'assiette contenant le spaghetti. Dans les deux cas, ces conditions sont respectées, ce qui confirme la validité de cet aspect des plans. Une contrainte supplémentaire est présente dans le scénario 10 puisque l'action doit être effectuée à un emplacement précis, soit sur le sofa. Pour ce faire, une fois les objets nécessaires à l'action obtenus, l'acteur se déplace à nouveau avant de s'asseoir sur le sofa, pour ensuite commencer l'action de manger. La scène animée est donc cohérente et les plans peuvent être déclarés valides.

Le scénario 15 est particulier car il est le seul à impliquer l'utilisation d'un liquide dans une scène. L'action à effectuer par l'acteur est de boire de l'eau. Cependant, le défi dans ce cas est l'obtention de cette eau, car le verre présent dans la scène est initialement vide et qu'aucune fontaine ne permet d'y boire de l'eau directement. Afin que la contrainte soit respectée, l'eau doit d'abord être obtenue, puis bue. Les actions présentes dans le plan généré consistent à se déplacer pour prendre le verre, se déplacer à nouveau près du robinet afin de verser l'eau à l'intérieur du verre et finalement de boire celle-ci par l'entremise du verre. Cette séquence d'action respecte la logique de la scène, qui ne permettait pas une autre séquence telle que, par exemple, verser l'eau avant d'obtenir le verre, ou traiter l'eau comme un objet et tenter de l'obtenir sans l'aide d'un contenant approprié. Le plan généré est donc valide et cohérent.

Enfin, le scénario 16 implique l'action de laver la vaisselle. Celle-ci est modélisée comme l'action de laver un objet, cet objet spécifique étant la vaisselle. Le défi de ce scénario est que,

sans être spécifié dans la description de scène, l'endroit auquel l'action doit être effectuée, le lavabo, doit être déduit implicitement afin que la scène soit cohérente. Ainsi, le plan généré contient d'abord l'action de se déplacer pour prendre la vaisselle, mais implique de se déplacer à nouveau vers le lavabo avant de commencer le lavage. Le plan est donc considéré valide dans le domaine que nous avons défini. Bien que nous considérons que l'action pourrait être développée davantage, par exemple l'utilisation obligatoire d'eau, de savon et d'une brosse pour le lavage, l'action de notre domaine ne comportait pas ces préconditions. Nous discutons de cette amélioration possible dans le chapitre 8, concernant les travaux futurs de notre projet.

AffectingDirectedMotion

Des actions de type AffectingDirectedMotion sont représentées dans les scénarios 3, 4, 9 et 12. Ces dernières sont celles comportant le plus de contraintes possibles au niveau de la description de scène puisqu'elles comportent un élément affecté ainsi qu'une possible route ou direction de mouvement.

Le scénario 3 implique l'action de pousser un objet, possiblement trop gros pour être pris mais pas nécessairement, à un emplacement précis dans la scène. Cet emplacement constitue la destination de la route empruntée par l'acteur et l'objet. Dans le cas présent, l'objet affecté est une machine à laver et la destination est près de la sécheuse. Les actions présentes dans le plan généré consistent donc à se rendre à la machine à laver pour ensuite la pousser jusqu'à l'emplacement de la sécheuse. Le plan est donc plutôt simple et considéré comme valide. Par contre, nous remarquons dans un tel cas que l'intégration de données numériques dans les problèmes aurait aidé à préciser davantage la position imprécise que constitue « près de la sécheuse ». Cet aspect est discuté à la section 7.6.

Le scénario 4 comporte la même structure que le précédent, mais implique plutôt l'action de placer un objet. Cet objet est une balle dont la destination est l'intérieur d'une boîte. L'action *place-in* de notre domaine remplit exactement cette tâche et se retrouve à la fin du plan généré. Cependant, afin d'être réaliste, l'acteur doit d'abord se déplacer pour prendre la balle et se rendre ensuite à l'emplacement de la boîte pour l'y placer. Il n'y a donc aucun problème puisque le plan indique exactement cette séquence d'actions.

Le scénario 9 impose des contraintes supplémentaires aux deux scénarios précédents puisqu'il implique l'action d'amener un objet d'un endroit à un autre. La description de scène spécifie donc une route comportant une source et une destination, tandis que l'action d'amener implique que l'acteur doit être en possession de l'objet du début à la fin de la route. Dans le cas présent, l'objet est un livre pris d'une étagère après le déplacement nécessaire pour s'y rendre, pour ensuite être amené jusqu'au lit grâce à un nouveau déplacement de l'acteur l'ayant en sa possession. La dernière action du plan indique que l'acteur se couche sur le lit

afin de respecter la contrainte que l'acteur doit terminer sa route sur le lit. Le plan suit donc la logique de l'énoncé initial et est valide.

Enfin, le scénario 12 implique l'action de lancer une balle vers le mur. Cette situation est modélisée par le *frame* « Cause-motion » et l'attribut *motionDirection* ayant comme valeur le mur. Le fait que le même *frame* soit utilisé que pour l'action de pousser crée une situation d'ambiguïté possible entre les deux actions. Par contre, la spécification d'un attribut de type *motionDirection* plutôt que de type *route* indique que l'acteur ne doit pas garder le contrôle de l'objet tout au long de son trajet, mais seulement assurer sa direction initiale. Les actions présentes dans la plan généré consistent d'abord à se déplacer pour prendre la balle, suivi de l'action de viser le mur et du lancer de la balle dans cette direction. De cette façon, l'acteur assure que la direction initiale du trajet de la balle soit le mur. Le plan est donc valide.

AffectingOrientationChange

Le dernier type d'actions présent dans les scénarios de test est AffectingOrientation-Change, représenté uniquement dans le scénario 6 dû au faible nombre de *frames* y étant associés (voir section 6.6.2). L'action impliquée dans ce scénario est celle de viser une autre personne avec une caméra. Cette situation est modélisée presque identiquement à celle du scénario 12 expliqué ci-haut, à l'exception que l'attribut de direction est de type *orientation-Direction* plutôt que *motionDirection* puisqu'il ne s'agit pas d'engendrer un mouvement de l'objet. Il est donc normal de retrouver la structure de la première partie du plan 12 dans le plan 6. L'acteur se déplace pour prendre la caméra et ensuite vise l'autre personne avec celle-ci. Ce plan est donc cohérent avec notre domaine d'actions.

7.3 Impact d'utilisation du format de représentation conceptuelle abstraite

Après l'obtention des résultats de la section 7.1 et l'analyse des ces derniers effectuée à la section 7.2, nous sommes en mesure d'affirmer qu'il est possible de concevoir un format de représentation conceptuelle abstraite contenant les informations essentielles à la création d'une scène animée. En effet, nous avons été en mesure de générer des plans valides à partir de situations exprimées dans notre format de représentation. Nous discutons ici de l'impact d'utilisation de ce dernier pour la formulation d'un problème de planification.

7.3.1 Structure du format

Le choix de la structure de notre format de représentation conceptuelle abstraite s'est avéré être une bonne décision, surtout en basant celui-ci sur le formalisme de *GUM-Space*.

Ce formalisme permet clairement de définir une situation en identifiant ses éléments principaux et en les associant à des rôles précis. Il permet ainsi de standardiser la description des scènes et donc de faciliter grandement le traitement de ces dernières. Notre format permet également la spécification de plusieurs configurations simultanées, donc d'une scène comportant plusieurs actions effectuées par un ou plusieurs acteurs. Nous n'avons cependant pas effectué de tests pour valider ce mécanisme dû aux contraintes supplémentaires que cela imposait, mais notre formalisme le permet. Quelques considérations concernant la définition d'une scène comportant plusieurs configurations sont énoncées à la section 7.7.1.

Grâce à une combinaison de plusieurs aspects de notre format, nous sommes en mesure de catégoriser et traiter spécifiquement les types d'actions que nous avons identifiés, permettant ainsi un traitement adapté au type de situation que nous désirons animer. En effet, la catégorisation des types de configurations jumelée au schéma de validation permet d'identifier les attributs précis requis pour chacun de ces types et ainsi limiter le nombre d'incohérences possibles en entrée. Par exemple, il est impossible de spécifier une situation dont la configuration est de type AffectingSimpleMotion et possédant un attribut *route*. Ceci augmente la robustesse de notre solution face à des erreurs de modélisation et de description de la scène.

7.3.2 Liens vers les ressource sémantiques et la base de connaissances

Outre la structure de notre format de représentation, l'aspect principal de ce dernier ayant un impact positif dans notre solution est l'intégration des ressources sémantiques et de la base de connaissances. Si la structure de la description de scène permet d'identifier clairement les rôles de chaque élément, l'intégration de la connaissance permet d'identifier concrètement les types de chacun d'eux. Cela permet, lors de la phase de traitement de la scène, d'associer chaque élément abstrait de la description de scène tels des objets ou modalités spatiales à un objet ou prédicat concret du problème de planification formulé.

Notre format de représentation permet également l'identification des actions de la scène par l'entremise de l'utilisation des ressources sémantiques *GUM* et *FrameNet*. L'impact d'utilisation de ces dernières est discuté à la section 7.4.

7.3.3 Format de représentation des objets de la scène statique

En plus de notre format de représentation conceptuelle abstraite, notre méthodologie a permis d'évaluer l'apport de notre format de représentation des objets de la scène statique. D'abord, ce dernier format dérivant du premier, il hérite des avantages de l'utilisation de liens vers la base de connaissances ainsi que de des relations spatiales de la ressource *GUM-Space*. Puis, suite à l'analyse de résultats de la section 7.2, nous pouvons affirmer que notre format

de représentation des objets de la scène statique contient suffisamment d'information pour permettre la modélisation du problème de planification sans erreur de cohérence dans le plan généré.

Ce format a été conçu dans l'optique de fournir l'information minimale concernant les objets présents dans la scène statique tout en facilitant le processus de traitement des informations d'entrée. Il est clair pour nous qu'il s'agit là d'une représentation intermédiaire et que l'idéal serait l'utilisation du format réel représentant la scène statique avec les termes numériques de positions et les informations sémantiques des objets, puisqu'il s'agit de la représentation exacte de la scène statique. Par contre, cette approche compliquerait de beaucoup le processus de traitement de l'information, tout en forçant la déduction de certaines informations qui ne seraient pas nécessairement présentes dans le format réel, telles les relations de contrôle.

7.4 Impact des ressources sémantiques utilisées

La seconde partie de la validation de l'hypothèse H1 de notre problématique consiste à calculer l'impact de l'utilisation des ressources sémantiques dans notre solution. Les ressources que nous avons utilisées sont *FrameNet* pour la classification précise des actions et *GUM/GUM-Space* pour la catégorisation générale des actions, la description de situations grâce au concept de configuration et des ses attributs ainsi que pour l'expression de modalités spatiales. Nous discutons ici de l'apport de ces deux ressources dans notre solution, au niveau du format de représentation de la scène.

7.4.1 *GUM/GUM-Space*

L'utilisation de l'ontologie *GUM-Space* pour la représentation de plusieurs éléments de notre format s'est avérée un choix judicieux. En effet, presque l'entièreté de la structure et des éléments présents dans notre format dérivent de concepts définis dans cette ressource.

D'abord, comme mentionné à la section 3.3.1, les configurations de *GUM* sont divisées en trois types différents. Cette division nous a permis de définir le domaine des situations que nous souhaitions traiter, c'est-à-dire celles correspondant aux configurations de type *Doing and Happening*. Nous avons ensuite décidé d'utiliser la classification générale des actions définie par *GUM-Space* pour ce type de configuration. Cela nous a permis la paramétrisation selon les attributs des types de configurations permises dans notre format et donc la définition de notre schéma de validation.

Le fait que l'ontologie *GUM-Space* soit axée sur la représentation sémantique de l'information fonctionnelle et spatiale est un avantage pour notre projet car le but de représentation

de l'information est le même. L'utilisation des attributs et modalités spatiales définis par celle-ci est donc également un avantage important, puisqu'ils ont permis de représenter toute l'information spatiale contenue dans une situation à l'aide d'attributs et de concepts précis dans notre représentation conceptuelle abstraite. De plus, la motivation initiale de cette ontologie étant la présentation d'information provenant d'une phrase, la philosophie derrière sa conception rejoint celle du projet GITAN et pourrait éventuellement occuper un rôle encore plus important pour la traduction de l'information textuelle vers une animation.

7.4.2 *FrameNet*

L'utilisation de *FrameNet* pour la représentation abstraite des actions de la scène s'est également avérée un choix intéressant, mais loin d'être parfait. En effet, la différence de philosophie entre notre projet et *FrameNet* est ressortie à quelques reprises lorsque nous tentions de les intégrer ensemble. Par contre, l'apport de *FrameNet* à notre solution est malgré tout considérable.

D'abord, *FrameNet* se distingue des autres ressources linguistiques de classification des actions du monde réel par l'origine de sa classification, qui repose sur une représentation sémantique de l'action plutôt que sur sa racine textuelle. L'utilisation de cette ressource nous a permis de remplir un besoin fondamental de notre format de représentation, c'est-à-dire la description de l'action, tout en permettant une abstraction totale face à une représentation textuelle de la situation. Combinée aux concepts de l'ontologie *GUM-Space*, les *frames* de *FrameNet* nous ont permis d'identifier clairement l'action en cours dans la description de scène et ainsi permettre l'identification des buts à atteindre lors de la phase de formulation du problème de planification.

Cependant, l'utilisation de la ressource *FrameNet* n'est pas optimale pour notre projet, dû à la granularité souvent trop faible de certains *frames*. En effet, puisque *FrameNet* est avant tout une ressource linguistique, les *frames* ne tiennent pas compte de la représentation spatiale des actions définies, ce qui résulte parfois en une généralisation de plusieurs concepts qui sont distincts pour une animation 3D. Par exemple, tout type de mouvement du corps est représenté par le *frame* « Self-motion », qu'il s'agisse de l'action de marcher, danser ou courir. Un second exemple concerne le *frame* « Ingestion », qui regroupe les actions concrètes de manger et boire. Parmi les *frames* incluses dans nos scénarios de tests, « Change-posture » et « Cause-motion » posent également ce type d'ambiguïtés.

Cette légère différence dans la granularité des actions à représenter est parfois palliable par simple déduction à partir des attributs de l'action, mais peut également être indiscernable à quelques occasions, par exemple entre l'action de courir et de danser. Ceci annule la possibilité d'une association directe entre les *frames* de *FrameNet* et des ensembles de

buts paramétrables pour le problème de planification formulé. Cette dernière possibilité, très intéressante du point de vue de l'abstraction des connaissances, a été initialement envisagée pour notre solution, mais a dû être écartée pour les raisons mentionnées.

7.5 Impact du format de représentation des connaissances

Une des parties les plus importantes de notre projet constitue la conception et l'intégration d'une base de connaissances afin de permettre l'ajout d'information sémantique à nos scènes virtuelles. Suite à l'analyse des résultats de la section 7.2, nous sommes en mesure d'évaluer notre hypothèse H2 et d'affirmer qu'il est possible d'obtenir l'information nécessaire à la formulation d'un problème de planification à partir d'une base de connaissances. Nous discutons ici de l'impact de notre ontologie dans la réalisation de notre solution.

7.5.1 Hiérarchie de classes

L'apport principal de notre ontologie pour la formulation du problème de planification est la définition des hiérarchies de classes d'objets et de contrôleurs. Nous avons été en mesure de définir ces hiérarchies axées sur les fonctionnalités des objets, c'est-à-dire que la liste des fonctionnalités offertes par un objet est égale à celle de l'ensemble de ses classes parentes. Cette décision de conception s'est avérée payante lors de la résolution du problème de planification, puisque le polymorphisme des types d'objets est supporté par le langage de planification. Par exemple, dans la situation décrite à la section 5.4, l'objet *plate* est associé à sept types différents, ce qui lui permet d'hériter de fonctionnalités associées à plusieurs de ces types au cours du déroulement du plan d'animation. La classe *Container* lui permet d'abord d'être spécifié comme contenant dans la situation initiale. Puis, la classe *Object* lui permet de servir d'objet préhensible par l'acteur. Enfin, la classe *Tableware* lui permet de servir de contenant dans lequel peut être mangé le spaghetti.

Par contre, sur le plan de la conception des hiérarchies de classes, cette approche n'est pas toujours conviviale. En effet, la définition des fonctionnalités d'objets selon des classes parentes mène parfois à une sur-catégorisation des objets qui souvent se chevauchent entre elles. Par exemple, il est possible de diviser la classe *Food* selon sa nature (fruit, légume, viande, etc.), selon l'ustensile nécessaire pour la consommer (cuillère, fourchette, sans ustensile) ou selon la température à laquelle elle est normalement consommée (froid, chaud, température pièce). Conformément à notre approche de classification, chacune de ces options serait une sous-classe de la classe *Food* et chaque objet constituant de la nourriture devrait se souscrire à un minimum de trois classes parentes uniquement pour cette catégorie. Cette approche est donc très efficace sur le plan de l'identification des fonctionnalités des objets,

mais alourdit considérablement la conception de l'ontologie. Considérant que notre ontologie n'implémente qu'une infime partie du domaine d'objet du monde réel, une ontologie complète serait extrêmement volumineuse.

7.5.2 Représentation d'information supplémentaire

Comme le mentionne l'objectif O4 de notre problématique, notre format de représentation des connaissances doit être compatible avec le projet parallèle de génération de scène statique. Dans cette optique, nous sommes en mesure d'affirmer qu'une ontologie a été un bon choix pour la représentation de la connaissance, puisqu'en plus de permettre l'implémentation d'une hiérarchie de classes complexe, elle permet l'expression de plusieurs types de propriétés et d'attributs utiles au projet parallèle, par exemple les objets nécessaires à la présence d'un autre objet dans la scène, ou les intervalles normaux de dimensions des objets.

Cependant, malgré la grande quantité d'information possible d'y représenter, notre idée initiale était de pouvoir représenter toute l'information du domaine grâce à notre ontologie, c'est-à-dire y inclure également les données du domaine d'actions. Nous avons dû nous rendre à l'évidence que le formalisme du langage OWL n'était pas adapté à ce type d'information. La raison principale derrière cette incompatibilité réside dans le fait que les informations représentées dans une ontologie peuvent être décomposées en énoncés sémantiques atomiques, ce qui est impossible pour les actions définies en PDDL. Par exemple, la définition de l'action *take* présentée à la figure 5.12 de la section 5.3.2 implique des équivalences entre les objets spécifiés en paramètres et ceux présents dans les listes de préconditions et d'effets. Le paramètre *human*, par exemple, est présent dans quatre des prédicats impliqués par cette action (*at, holding, has-taken, has-taken-from*). Il est donc impossible de diviser ces quatre prédicats en énoncés atomiques indépendants puisque la définition des paramètres les rend dépendants les uns des autres. Nous avons alors préféré conserver ces informations dans leur formalisme naturel.

7.6 Impact d'utilisation de la planification classique

L'ensemble de notre hypothèse de recherche globale repose sur le principe qu'il est possible d'obtenir un plan d'animation par planification classique. Nous avons ensuite formulé nos objectifs de recherche afin de parvenir à formuler un problème de planification résoluble pour obtenir ce plan. Nous discutons ici de l'impact d'utilisation de la planification classique pour la résolution du problème ainsi que des limitations posées par cette approche.

7.6.1 Résolution du problème de planification

Suite à l'obtention des résultats présentés à la section 7.1, nous pouvons affirmer d'emblée que la formulation d'un problème de planification résoluble permet directement la résolution de ce dernier. L'utilisation du planificateur Blackbox pour la résolution des problèmes de planification a été un choix convenable, compte tenu du fait que nous avons été en mesure d'obtenir tous les plans pour lesquels nous avions formulé un problème. Malgré l'absence de validation concrète pour cet énoncé, nous pouvons remarquer que l'hypothèse prononcée à la section 4.4 n'a pas été contredite. En effet, puisque le plus long plan que nous avons obtenu à partir des scénarios de test comporte 13 actions, le défi principal de la résolution des problèmes concerne l'étendue du domaine plutôt que la longueur des plans, surtout dans l'optique ou le domaine vise à être enrichi considérablement. Le choix d'un planificateur exploitant un algorithme de satisfaction de contraintes combiné à GRAPHPLAN s'est donc trouvé justifié dans notre contexte.

7.6.2 Limitations de la planification classique

Bien qu'il s'agisse de l'avenue que nous avons choisie dès le début de projet, l'utilisation de la planification classique pose quelques limitations de représentation de l'information et d'implémentation.

D'abord, l'utilisation de la planification classique implique le fait que l'état courant du monde doit être représenté par une conjonction de prédicats. Par contre, le monde que nous tenons à représenter est une scène virtuelle en trois dimensions. Évidemment, une transposition de l'ensemble des données d'une scène virtuelle vers une conjonction de prédicats atomiques implique une marge d'erreur substantielle.

La première distinction importante entre les deux représentations est l'abstraction des données numériques de la scène. Les objets sont représentés par des déclarations unitaires plutôt que par des volumes englobants ou des modèles. Cependant, cette abstraction est souhaitable puisque notre approche se concentre sur les fonctionnalités des objets en place plutôt que sur les mécanismes physiques d'interaction. La difficulté réelle de cette abstraction des données numériques constitue l'expression des emplacements spatiaux dans la scène. L'attribution de positions précises aux éléments de la scène est important pour le réalisme de cette dernière, mais ces informations ne sont pas représentées par les prédicats de l'état courant. L'ajout du prédicat *at* à notre domaine permet d'exprimer la position relative de l'acteur par rapport à un objet de la scène, mais pour certaines situations, il serait intéressant de posséder plus d'information sur les emplacements des objets, par exemple dans le cas où deux objets sont adjacents et ne requièrent pas de déplacement de l'acteur pour aller de l'un

à l'autre.

Une seconde distinction entre les deux représentations concerne le fait que dans une modélisation de problème en langage PDDL sous formalisme STRIPS, il n'est pas possible de créer ou détruire des objets en cours d'exécution. Nous entendons par création et destruction n'importe quelle action modifiant la nature d'un objet de la scène. Par exemple, l'action de manger un aliment en cause la destruction et l'action de fabriquer un objet en engendre la création. Ce problème peut sembler simple à contourner à la base, mais il implique que tout objet créé ou détruit au cours du déroulement d'une animation doit être planifié à l'avance, car celui-ci doit se retrouver dans la déclaration initiale des objets pour pouvoir exister dans la scène à n'importe quel moment. Pour les besoins de notre projet, nous avons écarté ce mécanisme de nos scénarios, mais dans l'optique future où le domaine d'actions serait de plus grande envergure (voir chapitre 8), ce problème devra être adressé.

Enfin, la représentation du monde est directement dépendante du formalisme accepté par le langage choisi. En effet, notre choix d'utiliser le langage PDDL sous formalisme STRIPS implique que notre représentation du monde doive se conformer aux limitations du langage. Nous avons déjà expliqué ce choix à la section 5.3.2, mais l'implémentation du même domaine en exploitant un langage plus permissif comme ADL, par exemple, serait une occasion intéressante de comparer les avantages et inconvénients des deux approches et de déterminer laquelle serait la plus efficace.

7.7 Limitations de la méthodologie d'évaluation

Dans le but de démontrer la faisabilité de notre approche de génération automatique de plans d'animations, nous avons élaboré la méthodologie d'évaluation présentée au chapitre 6. Cependant, cette dernière ne pouvait couvrir exhaustivement tous les cas envisageables en entrée. Nous avons donc dû limiter l'application de notre méthodologie à une partie du domaine. Nous discutons ici des limitations posées par notre méthodologie d'évaluation ainsi que des répercussions de celles-ci sur les conclusions à tirer à partir de notre expérience.

7.7.1 Contraintes de l'expérience

À la section 6.4.1, nous indiquons les contraintes auxquelles a été soumise notre expérience. Ces contraintes ont pour conséquence une certaine limitation dans l'application de notre méthodologie d'évaluation.

D'abord, nous avons limité les scénarios de tests à une seule action par description de scène. Puisque, par définition dans notre domaine, chaque action ne peut posséder qu'un seul acteur, nous limitons donc les scènes à un seul acteur également. Par contre, le format

de représentation conceptuelle abstraite de la scène, tel que nous l'avons défini, supporte la présence de plusieurs configurations, représentant chacune une action avec ses paramètres. Ces actions pourraient être traitées comme une liste séquentielle, pour laquelle seraient générés le même nombre de plans, en tenant compte de l'état final du plan précédant dans l'état initial de chaque plan, pour en arriver à un plan global regroupant plusieurs actions. Dans le cas d'actions simultanées, la résolution du problème de planification pourrait s'effectuer sans problème pour toutes les actions, mais notre méthode de traitement de la description de scène nécessiterait quelques ajustements importants afin de formuler correctement le problème, notamment pour inclure la possibilité de plusieurs acteurs dans le même scène.

Ensuite, une seconde limitation de notre méthodologie d'évaluation reliée aux contraintes de notre expérience concerne la quantité d'attributs traités dans nos scénarios de test. En effet, quelques attributs d'actions définis dans notre schéma de validation n'ont pas été couverts par nos scénarios de test. Vu la quantité de scénarios plutôt limitée, nous avons choisi de ne pas implémenter de scénario de test incluant les attributs *direction* et *orientationRoute*. Par contre, comme les attributs *route*, *motionDirection* et *orientationDirection* ont été implémentés avec succès et que ces derniers présentent une structure identique aux premiers, nous croyons pouvoir affirmer qu'ils sont compatibles avec notre solution. Le seul attribut pouvant nécessiter un effort supplémentaire est *pathPlacement*. En effet, ce dernier représente un chemin, c'est-à-dire un emplacement auquel l'acteur doit se rendre à un moment précis de sa route. Puisque la technique de planification classique que nous utilisons est axée sur les buts plutôt que sur le déroulement de la scène, il est difficile de formuler un but représentant un moment précis dans l'animation autre que la fin. Par contre, avec notre approche de définition de prédicats d'actions effectuées (voir section 5.3.2), il serait possible de spécifier ce chemin dans les buts en indiquant les déplacements précis à l'aide de ces prédicats. Cela ajouterait cependant une rigidité non souhaitée dans le déroulement de la scène, puisqu'il faudrait forcer l'exécution d'une action spécifique autre que celle spécifiée dans la description de scène.

Finalement, l'expression des modalités spatiales dans nos scénarios de test est basée sur celles définies par l'ontologie *GUM-Space*, qui comporte une cinquantaine de modalités spatiales différentes. Cependant, puisque l'ontologie est avant tout destinée à la représentation sémantique de phrases, certaines de ces modalités spatiales sont difficiles voire impossible à représenter dans une scène car elles concernent de l'information spatiale abstraite. Par exemple, les modalités *EastProjectionInternal* ou *RelativeNonProjectionAxial* sont définies pour représenter une information textuelle précise, mais peuvent très difficilement se représenter avec notre approche de modélisation de scène virtuelle. Dans notre expérience, nous avons ciblé quatre des modalités spatiales les plus souvent utilisées dans les situations proposées par

GUM-Space. Celles-ci sont les relations de contrôle *Containment* et *Support*, l'indicateur de direction *GeneralDirectional* ainsi que l'indicateur de proximité *Proximal*. Il serait possible d'en intégrer quelques autres telles que *DenialOfFunctionalControl* ou *Connection*, mais la majorité d'entre elles sont trop spécifiques ou non adaptées à notre approche de modélisation pour tenter des les représenter dans notre solution.

7.7.2 Validation des résultats

La méthode de validation des résultats que nous proposons à la section 6.6.1 consiste en une évaluation manuelle exhaustive des plans générés. Nous sommes donc positifs sur le fait qu'il s'agit d'une méthode suffisamment rigoureuse pour assurer la justesse de l'analyse de nos résultats et de notre discussion. Cependant, nous croyons que cette méthode d'évaluation pourrait être améliorée en deux points.

D'abord, un léger biais d'expérimentation peut être introduit dû au fait que la conception du programme, la création des scénarios de test et la validation des plans générés ont été effectuées par la même personne. Sans que cette dernière puisse être la cause d'erreurs volontairement introduites dans les résultats, la création des scénarios de test et la validation des plans générés auraient gagné en rigueur s'ils avaient été effectuées par une tierce personne. Faute de ressources nécessaires, nous avons procédé à l'exécution de ces tâches en bonne foi.

Ensuite, sans être un défaut de notre méthode d'évaluation, nous croyons que la possibilité de visualiser les scènes animées résultantes aurait été un avantage pour notre méthodologie. Par contre, cela requiert un travail important de combinaison du plan d'animation avec la scène statique initiale. Cette étape future du projet GITAN est discutée au chapitre 8.

7.8 Analyse de la couverture du domaine d'actions

La dernière partie de notre discussion concerne l'analyse de la couverture du domaine d'actions. Le but de cette analyse est de calculer la proportion des actions du domaine couvertes par notre solution. Nous utilisons les *frames* de la ressource *FrameNet*, dont nous avons présenté une classification à la section 6.6.2.

7.8.1 Catégories d'actions couvertes

D'abord, parmi les onze catégories que nous avons identifiées, nous avons choisi d'implémenter des scénarios pour cinq d'entre elles. En effet, nous avons couvert des actions provenant des catégories NonAffectingSimpleMotion, NonAffectingDirectedMotion, AffectingSimpleMotion, AffectingDirectedMotion et AffectingOrientationChange. Ces catégories seront traitées à la section 7.8.2.

Parmi les catégories non couvertes, une seule constitue une catégorie basée sur celles de l'ontologie *GUM-Space*. Il s'agit de la catégorie NonAffectingOrientationChange. Les deux *frames* contenus dans celle-ci concernent, comme le nom l'indique, les actions de changement de direction de l'acteur dans la scène. Puisque notre approche de modélisation de la scène ne comporte aucun indicateur de direction de l'acteur, il n'est pas possible de représenter ces actions.

Les cinq autres catégories représentent des actions que nous n'avons pas pu associer à la classification de *GUM-Space*, donc que nous ne souhaitons pas traiter avec notre solution. Dans l'optique ou *FrameNet* est avant tout une ressource linguistique, il est clair qu'une partie de son contenu n'est pas compatible avec une représentation spatiale dans une scène virtuelle. Les actions sensorielles, morales et d'état en font partie, puisqu'elles représentent majoritairement des actions n'ayant aucune représentation visuelle dans l'espace. D'un autre côté, les actions de l'espace demandent une vision d'ensemble de la scène pour en comprendre le sens par une animation, ce que nous ne pouvons représenter avec notre approche de modélisation. Finalement, les actions complexes représentent des processus trop complexes pour la représentation par une simple animation.

7.8.2 Proportion d'actions couvertes

Une fois ce premier tri effectué, nous nous concentrons sur les 59 *frames* contenus dans les cinq catégories qui nous intéressent. Puisque notre projet ne vise qu'à fournir une preuve de concept, nous n'avons évidemment pas implémenté des scénarios de test pour tous ces frames. Cependant, grâce à notre catégorisation de ceux-ci, nous sommes en mesure d'affirmer qu'il serait possible de les représenter, car ils peuvent se représenter avec les mêmes paramètres que les *frames* que nous avons testés.

Il y a toutefois quelques réserves liées au limitations de notre solution, qui sont mentionnées dans la discussion de sections 7.3 à 7.6. Nous abordons ici les proportions des *frames* que nous considérons couverts par notre solution.

Pour la catégorie NonAffectingSimpleMotion, nous avons testé les 2 *frames* avec succès lors de notre expérimentation. Le taux de couverture est donc de 100%.

Pour la catégorie NonAffectingDirectedMotion, nous avons testé 3 des 7 *frames* avec succès lors de notre expérimentation. Nous considérons que tous les *frames* de la catégorie sont représentables par les mêmes paramètres, à l'exception de *Motion-directional*, qui constitue une action passive. Nous évaluons donc le taux de couverture à 86%.

Pour la catégorie AffectingSimpleMotion, nous avons testé 2 des 17 *frames* avec succès lors de notre expérimentation. Nous considérons que tous les *frames* de la catégorie sont représentables par les même paramètres, à l'exception de *Cause-to-fragment*, *Cutting*, *Destroying* et

Grinding, qui constituent des actions de division physique ou de destruction d'objets. Nous évaluons donc le taux de couverture à 76%.

Pour la catégorie AffectingDirectedMotion, nous avons testé 3 des 31 *frames* avec succès lors de notre expérimentation. Nous considérons que tous les *frames* de la catégorie sont représentables par les mêmes paramètres, à l'exception de *Cooking-creation*, qui constitue une action de création d'objets. Nous évaluons donc le taux de couverture à 97%.

Pour la catégorie AffectingOrientationChange, nous avons testé 1 des 2 *frames* avec succès lors de notre expérimentation. Nous considérons que le *frame Cause-to-move-in-place* n'est pas représentable par les mêmes paramètres car il implique l'orientation des objets de la scène, qui nous est inconnue dans notre modélisation. Nous évaluons donc le taux de couverture à 50%.

Le résultat global de notre analyse de couverture du domaine d'actions donne une couverture potentielle de 52 *frames* sur 59, pour un taux de couverture de 88%. Nous croyons que ce taux est suffisamment élevé pour que notre approche soit considérée intéressante et que des travaux futurs visent à l'expansion de notre preuve de concept vers un système à part entière. Nous discutons de ces travaux futurs au chapitre 8.

7.9 Atteinte des objectifs de recherche

Au terme de l'analyse des résultats de notre expérimentation, nous sommes en mesure d'affirmer que nous avons atteint nos objectifs de recherche. En effet, nous avons été en mesure d'identifier les informations essentielles à la création d'une scène animée, pour ensuite concevoir un format de représentation conceptuelle abstraite de scène basé en partie sur l'ontologie *GUM-Space* et un format de représentation des objets de la scène statique. Nous avons également été en mesure de concevoir une base de connaissances axée sur la fonctionnalité des éléments de la scène et compatible avec le projet parallèle de génération de scène statique. Puis, nous avons pu implémenter un programme capable de traiter l'information des différentes sources mentionnées ci-haut pour en formuler un problème de planification classique et ensuite en assurer la résolution grâce à un planificateur. Finalement, afin de valider notre approche, nous avons démontré l'applicabilité de notre solution à une partie importante du domaine de conceptualisation des actions du monde réel.

L'atteinte de nos objectifs de recherche nous amène à répondre à nos hypothèses de recherche et à notre question de recherche. Nous pouvons d'abord affirmer qu'il est possible de concevoir un format de représentation conceptuelle abstraite contenant les informations essentielles à la création d'une scène animée. Nous sommes également en mesure d'affirmer qu'il est possible d'obtenir l'information nécessaire à la formulation d'un problème de planifi-

cation à partir d'une base de connaissances. Enfin, nous pouvons affirmer qu'il est possible de formuler un problème de planification classique résoluble à partir de la représentation conceptuelle abstraite de la scène animée. Considérant la confirmation de nos trois hypothèses de recherche par le biais de l'atteinte de nos objectifs, nous pouvons répondre à notre question de recherche et conclure qu'il est possible de générer automatiquement un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite.

À la suite de l'analyse de nos résultats, nous sommes également en mesure d'évaluer les orientations à prendre pour la suite des travaux, autant pour la génération améliorée de plans d'animation que pour l'ensemble du projet GITAN. Nous discutons de ces orientations au chapitre 8.

CHAPITRE 8

TRAVAUX FUTURS

Ce chapitre a pour but de présenter les orientations futures des travaux visant à améliorer la génération automatique de plans d'animations à partir d'une représentation conceptuelles abstraites. Nous discutons d'abord du développement d'une ontologie et d'un domaine d'actions de grande envergure, pour ensuite discuter de la génération automatique d'animations à partir de texte.

8.1 Développement d'une ontologie de grande envergure

Comme nous l'avons mentionné à quelques reprises lors de notre discussion du chapitre 7, la portée de notre projet était de servir de preuve de concept pour la génération automatique de plans d'animations. Une fois notre approche validée par la preuve de concept, l'une des avenues possibles pour des travaux futurs est de transformer notre preuve de concept en un système à part entière. Pour ce faire, la base de connaissances doit être développée à une échelle beaucoup plus grande. Cela signifie que cette dernière soit adaptée ou complètement reconçue, dans le but de couvrir l'ensemble des objets du monde, ou du moins l'ensemble des objets que l'on souhaite représenter dans une scène virtuelle.

De plus, dans la philosophie de la conception d'ontologies est inclus le partage d'information, c'est-à-dire la mise en ligne de nos données et l'ajout de liens vers des ontologies existantes. Dans l'optique de conception d'une ontologie de grande envergure, celle-ci gagnerait à inclure des liens vers des ontologies pouvant ajouter de l'information pertinente à notre modèle. Par exemple, une ontologie de modélisation des relations spatiales telle que *GUM-Space* (Bateman *et al.* (2010)) pourrait être liée aux types de contrôleurs de notre ontologie, tandis que la définition des actions de notre domaine dans l'ontologie pourrait bénéficier de l'information de sens commun contenue dans la ressource ConceptNet (Liu et Singh (2004b)). Nous croyons toutefois que notre ontologie doit demeurer une entité à part entière plutôt qu'une extension d'une ontologie existante, car le biais introduit par le domaine d'application différent des autres ontologies serait important pour une application aussi spécifique que la nôtre.

8.2 Développement d'un domaine d'actions de grande envergure

Dans la même optique que celle mentionnée à la section 8.1, c'est-à-dire de transformer notre preuve de concept en système à part entière, le domaine d'actions doit également être développé à une grande échelle. Cela signifie que celui-ci soit enrichi de nombreuses nouvelles actions, mais aussi que les énoncés reliés à ces actions soient améliorés.

Le développement d'un domaine d'actions de grande envergure doit se faire en parallèle avec le développement de l'ontologie afin de s'assurer de la cohérence des types d'objets concernés par les différentes actions ainsi que des préconditions et effets de ces dernières. Parmi les aspects de notre domaine d'actions qu'il serait intéressant de peaufiner, nous croyons que les listes de préconditions et effets des actions devraient être enrichies afin de répondre à la nouvelle envergure de l'ontologie des objets du monde. Nous croyons que le modèle d'actions tel que nous l'avons défini est valable, mais une expansion de celui-ci nécessiterait une plus grande précision quant au déroulement possible des actions dans des situations précises. Cette précision peut être obtenue en spécifiant davantage de préconditions nécessaires à l'exécution de chaque action, augmentant du coup le niveau de réalisme des animations générées. Un mécanisme intéressant pour arriver à cette fin serait l'inclusion d'instruments, c'est-à-dire des objets devant être en possession de l'acteur afin que l'exécution d'une action soit possible. Ce concept a été exploité à quelques endroits dans notre domaine d'actions, par exemple pour l'action de manger, sans toutefois être défini globalement pour une utilisation plus générale dans l'ensemble du domaine. Notre ontologie prévoit déjà une propriété permettant la spécification d'un instrument pour une action donnée, mais l'implémentation de ce mécanisme au niveau du domaine d'actions serait une avancée intéressante.

Dans un second ordre d'idées, nous croyons qu'il pourrait être intéressant d'étudier la possibilité de définir un modèle de représentation plus abstrait pour la description des actions du domaine. Malgré que l'ensemble de notre projet soit axé sur l'abstraction de l'information pour la génération d'un plan d'animation, nous avons décidé de recourir à une description directe des actions en langage PDDL, car la formalisation des paramètres, préconditions et effets n'était pas compatible avec le format de notre ontologie. Il serait donc intéressant de parvenir à exprimer ces énoncés dans un formalisme plus abstrait, qui les représenterait conceptuellement tout en étant indépendant du langage de description du problème de planification.

8.3 Génération automatique d'animations à partir de texte

Notre projet s'inscrit dans les travaux du projet GITAN, dont l'objectif global est la génération automatique d'animations 3D à partir de texte. Au terme de la réalisation de

notre expérience, nous croyons avoir contribué à l'avancement du projet GITAN par une approche valable de génération automatique de plans d'animations. Cela dit, beaucoup de travail reste à effectuer pour atteindre l'objectif global. Nous discutons ici de travaux futurs à entreprendre afin de compléter l'intégration de nos travaux dans le projet GITAN.

8.3.1 Analyse de texte vers une représentation conceptuelle abstraite

La représentation conceptuelle abstraite de la scène est le point de départ du traitement de l'information dans notre projet. Or, à ce jour, les travaux sur l'analyse syntaxique et sémantique de texte dans le projet GITAN produisent des résultats intéressants, mais ne sont pas encore aptes à traduire l'information contenue dans le texte vers notre format de représentation, basé sur le formalisme de GUM-Space. Le développement d'un analyseur sémantique capable d'effectuer une telle tâche est essentiel afin de permettre la liaison entre les travaux sur le traitement du langage naturel et notre projet.

8.3.2 Génération automatique de scènes statiques

Tel que mentionné à plusieurs reprises précédemment, un projet parallèle au nôtre et également inclus dans le projet GITAN vise à la génération automatique d'une scène statique à partir d'une représentation abstraite. Celui-ci constitue l'étape précédente par rapport à notre projet dans le processus de traitement de l'information contenue dans la description de scène. Comme mentionné à la section 5.2 de notre solution proposée, cette étape est essentielle à la génération d'un plan d'animation car elle définit le contenu de la scène qui sera utilisée pour l'animation. Au terme de ce projet, l'information conceptuelle abstraite contenue dans la description de scène aura été traitée par les deux modules pour produire une scène 3D ainsi qu'un plan d'animation à exécuter dans cette scène.

8.3.3 Application des plans d'animations aux scènes statiques

L'application de plans d'animations aux scènes statiques associées constitue l'étape directement suivante à notre projet. Le défi consiste à matérialiser l'exécution du plan d'animation par un acteur dans la scène virtuelle fournie par le module de génération de scène. Il s'agit d'un problème intéressant, considérant que le plan doit être appliqué le plus fidèlement possible tout en respectant les contraintes physiques posées par la scène virtuelle. De plus, un certain format de représentation de la connaissance devra être utilisé afin de contenir l'information relative à la représentation spatiale des actions du domaine ainsi que les mécanismes et contraintes physiques d'interaction avec les objets de la scène. Une fois ce travail effectué, la majeure partie du travail de génération d'une scène animée sera complété.

8.3.4 Intégration de modèles et d'animations 3D

Enfin, afin de compléter la production de la scène animée, la dernière perspective de travaux futurs que nous abordons est celle de l'intégration de modèles 3D dans la scène virtuelle produite par les étapes précédentes. Bien que cela puisse sembler trivial à première vue, une problématique de granularité des modèles s'impose, puisque cela semble impensable de vouloir conserver dans une base de données un modèle 3D complet pour chaque objet du monde à représenter, quel que soit cet objet. Une approche de classification des modèles selon la fonctionnalité des objets qu'ils représentent combinée à une étude de modification de la morphologie des objets pourraient être intéressantes à explorer afin de réduire le nombre de modèles à intégrer tout en offrant une couverture maximale des objets du monde. La question des cycles d'animation de l'acteur pour la représentation des différentes actions devra aussi être abordée, tout comme les mécanismes internes d'interaction avec les objets physiques de la scène. Cette étape du processus constitue un point crucial dans la génération d'une scène animée car la représentation physique des éléments et actions importe beaucoup pour la compréhension des scènes ainsi exprimées.

CHAPITRE 9

CONCLUSION

Le but de notre projet était de démontrer la faisabilité de la génération automatique d'un plan d'animation par planification classique à partir d'une représentation conceptuelle abstraite. À la suite de notre expérimentation, nous sommes en mesure d'affirmer que notre solution répond aux objectifs de recherche posés.

Notre solution est d'abord basée sur la définition d'un format XML de description de scène inspiré de concepts définis par l'ontologie *GUM-Space*. Ce format de représentation conceptuelle abstraite d'une scène animée est précisément défini grâce à un schéma de validation. Sa structure est basée sur celle du concept de *configuration* défini par *GUM-Space*, représentant une situation où se déroule une action précise. La représentation des actions est basée sur la ressource sémantique *Framenet* ainsi que sur la classification des actions de GUM-Space. Selon le type d'action, plusieurs attributs peuvent qualifier le déroulement de celle-ci en fonction des objets concernés où encore selon des modalités spatiales.

Afin d'ajouter l'information nécessaire à la formulation d'un problème de planification, nous avons conçu et développé une ontologie d'objets du monde réel et de contrôleurs en langage OWL. Cette dernière permet d'enrichir chaque situation décrite par de l'information concrète sur les fonctionnalités offertes par les objets de la scène. Elle fournit également de l'information utile au déroulement de l'action dans la scène. Cette base de connaissances est combinée à la définition d'un domaine d'actions en langage PDDL pour la résolution du problème de planification formulé à partir de la représentation conceptuelle abstraite.

Nous avons mis au point une méthodologie d'évaluation impliquant l'implémentation de 16 scénarios de tests divers que nous avons validé manuellement afin de vérifier la validité de notre solution. Notre expérimentation a permis l'obtention de plans d'animations, c'est-à-dire de séquences ordonnées d'actions ponctuelles ayant pour but l'exécution, dans la scène virtuelle donnée, de l'action initialement décrite. Suite à un examen rigoureux des résultats, nous avons pu confirmer la validité de ceux-ci dans le contexte de notre évaluation. Nous avons ensuite procédé à une analyse de la couverture du domaine d'actions du monde réel. Cette dernière a révélé des taux de couverture potentielle de 50% à 100% pour chacune des catégories d'actions auxquelles nous nous intéressions, avec un taux global de 88%.

Bien que nous puissions affirmer avoir atteint nos objectifs de recherche, nous avons identifié certaines limitations à notre solution. D'abord, l'utilisation de *FrameNet* comme ressource sémantique pour la classification des actions pose quelques limitations quant à la

granularité des actions que nous pouvons représenter. Malgré que cette ressource est d'une grande utilité pour la représentation conceptuelle des actions du monde réel, sa méthode de classification n'est pas optimale pour notre projet. De plus, nous avons également identifié quelques limitations reliées à l'utilisation de la planification classique. En effet, l'absence de données numériques ainsi que de la possibilité de créer ou détruire des objets de la scène en cours d'exécution a limité l'étendue de la couverture du domaine d'actions par notre solution.

Nous avons également identifié quelques améliorations possibles à la méthodologie d'évaluation que nous avons utilisée. Celle-ci gagnerait à inclure un plus grand nombre d'attributs ainsi que de modalités spatiales dans des scénarios de tests supplémentaires afin d'évaluer avec plus de précision l'ensemble des situations possiblement traitables par notre solution. De plus, il serait intéressant de tester l'inclusion de plusieurs configurations dans la même scène, c'est-à-dire le déroulement de plusieurs actions, simultanées ou séquentielles, dans une seule scène.

Les travaux futurs reliés à notre projet de recherche impliquent la transformation de notre preuve de concept en un système à part entière. Pour ce faire, le développement d'une ontologie de grande envergure ainsi que d'un domaine d'actions à plus grande échelle sont essentiels. Ces orientations peuvent par exemple inclure l'ajout de liens entre les concepts de notre ontologie et des ressources déjà existantes, ainsi que l'enrichissement de notre modèle de sens commun au niveau des paramètres, préconditions et effets des actions du domaine. De plus, nous croyons qu'il serait intéressant d'étudier la possibilité de définir un modèle de représentation plus abstrait pour la description de ces dernières.

Enfin, dans le cadre du projet GITAN visant la génération automatique d'animations 3D à partir de texte, notre solution combinée à la génération automatique de scènes statiques permet la traduction d'information conceptuelle abstraite en information concrète de scène virtuelle. Les travaux à entreprendre afin de compléter l'intégration de nos travaux dans le projet GITAN incluent le développement d'un analyseur sémantique permettant la traduction de l'information textuelle vers notre format de représentation, l'application des plans d'animations aux scènes statiques ainsi que l'intégration de modèles et d'animations 3D.

RÉFÉRENCES

ABACI, T., CIGER, J. et THALMANN, D. (2005). Planning with smart objects. *In The 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision : WSCG*. 25–28.

BAIER, J. A., BACCHUS, F. et MCILRAITH, S. A. (2009). A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173, 593–618.

BAKER, C. F., FILLMORE, C. J. et LOWE, J. B. (1998). The Berkeley FrameNet Project. *Proceedings of the 36th annual meeting on Association for Computational Linguistics* -, 86.

BATEMAN, J., HOIS, J., ROSS, R. et TENBRINK, T. (2010). A linguistic ontology of space for natural language processing. *Artificial Intelligence*, 174, 1027 – 1071.

BATEMAN, J. A., HENSCHEL, R. et RINALDI, F. (1995). The Generalized Upper Model 2.0. Rapport technique, GMD/Institut für Integrierte Publikations- und Informationssysteme, Darmstadt, Germany.

BATEMAN, J. A., KASPER, R. T., MOORE, J. D. et WHITNEY, R. A. (1990). A General Organization of Knowledge for Natural Language Processing : the Penman Upper Model. Rapport technique, USC/Information Sciences Institute, Marina del Rey, California.

BERNERS-LEE, T., HENDLER, J. et LASSILA, O. (2001). The semantic web. *Scientific American*, 284, 34–43.

BLUM, A. L. et FURST, M. L. (1997). Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90, 281–300.

BONET, B. (1983). HSP : Heuristic search planner. *Linguistics*, 21, 1–8.

BONET, B. et GEFFNER, H. (2001). Heuristic search planner 2.0. *AI Magazine*, 22, 77.

CARBONELL, J., ETZIONI, O., GIL, Y. et JOSEPH, R. (1991). Prodigy : An integrated architecture for planning and learning. *ACM SIGART*, 2, 51–55.

COYNE, B. et SPROAT, R. (2001). Wordseye : An automatic text-to-scene conversion system. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM New York, NY, USA, vol. ternetchap, 487–496.

DO, M. et KAMBHAMPATI, S. (2001). Planning as constraint satisfaction : Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132, 151–182.

EROL, K., HENDLER, J. et NAU, D. S. (1995). Semantics for Hierarchical Task-Network Planning. Rapport technique.

FIKES, R. E. et NILSSON, N. J. (1971). STRIPS : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189–208.

GUTIERREZ, M., VEXO, F. et THALMANN, D. (2005). Semantics-based representation of virtual environments. *International Journal of Computer Applications in Technology*, 23, 229–238.

HELMERT, M. (2006). The Fast Downward Planning System. *Artificial Intelligence*, 26, 191–246.

HOFFMANN, J. et NEBEL, B. (2001). The FF Planning System : Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14, 253–302.

HORRIDGE, M. et BECHHOFER, S. (2009). The OWL API : A Java API for Working with OWL 2 Ontologies. *Proceedings of the OWL : Experiences and Directions*. Chantilly, USA, vol. 2009.

IRAWATI, S., CALDERÓN, D. et KO, H. (2006). Spatial ontology for semantic integration in 3D multimodal interaction framework. *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications - VRCIA '06*, 1, 129.

KAELBLING, L. P., LITTMAN, M. L. et CASSANDRA, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 99–134.

KALLMANN, M. et THALMANN, D. (1998). Modeling objects for interaction tasks. *Proc. Eurographics Workshop on Animation and Simulation*. 73–86.

KAUTZ, H., MCALLESTER, D. et SELMAN, B. (1996). Encoding plans in propositional logic. *Principles of Knowledge Representation and Reasoning International Conference*. 374–385.

KAUTZ, H. et SELMAN, B. (1998). BLACKBOX : A new approach to the application of theorem proving to problem solving. *AIPS98 Workshop on Planning as Combinatorial Search*. Citeseer, vol. 58260.

KESSING, J., TUTENEL, T. et BIDARRA, R. (2009). Services in Game Worlds : A Semantic Approach to Improve Object Interaction. *Proceedings of the 8th International Conference on Entertainment Computing*. vol. 5709, 276–281.

LASSILA, O. et SWICK, R. R. (1998). Resource Description Framework (RDF) Model and Syntax Specification. Rapport technique October, W3C.

LENAT, D. B. (1995). CYC : a large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38, 33–38.

LIU, H. et SINGH, P. (2004a). Commonsense reasoning in and over natural language. *Knowledge-Based Intelligent Information and Engineering Systems*. Springer, 293–306.

LIU, H. et SINGH, P. (2004b). ConceptNet - A Practical Commonsense Reasoning Tool-Kit. *BT Technology Journal*, 22, 211–226.

MA, M. (2006). *Automatic conversion of natural language to 3D animation*. Thesis, University of Ulster, Faculty of Engineering.

MCDERMOTT, D., GHALLAB, M., HOWE, A., KNOBLOCK, C., RAM, A., MANUELA VELOSO, D. et WELD, D. W. (1998). PDDL - The Planning Domain Definition Language. Rapport technique, Yale Center for Computational Vision and Control.

MCGUINNESS, D. L. et VAN HARMELEN, F. (2004). OWL Web Ontology Language Overview. Rapport technique, W3C.

MILLER, G. A. (1995). WordNet : a lexical database for English. *Communications of the ACM*, 38, 39–41.

NILES, I. et PEASE, A. (2001a). Origins of The IEEE Standard Upper Ontology Strategy. *2001 Workshop on the IEEE Standard Upper Ontology*. Seattle, 37–42.

NILES, I. et PEASE, A. (2001b). Towards a Standard Upper Ontology. *Proceedings of the international conference on Formal Ontology in Information Systems*. 2–9.

OTTO, K. (2005). The semantics of multi-user virtual environments. *Proc. of the Workshop towards Semantic Virtual Environments*. Freie Universität Berlin, Institut für Informatik, 1–10.

PEDNAULT, E. P. D. (1989). ADL : Exploring the middle ground between STRIPS and the situation calculus. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. Toronto, Canada, 324–332.

PEDNAULT, E. P. D. (1994). ADL and the State-Transistion Model of Action. *Journal of Logic and Computation*, 4, 467–512.

PENBERTHY, J. S. et WELD, D. S. (1992). UCPOP : A Sound, Complete, Partial Order Planner for ADL. *Proceedings of the third international conference on knowledge representation and reasoning*. 103–114.

PETERS, C., MAC NAMEE, B., DOBBYN, S. et O'SULLIVAN, C. (2003). Smart objects for attentive agents. *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*.

PRUD'HOMMEAUX, E. et SEABORNE, A. (2008). SPARQL Query Language for RDF. Rapport technique, W3C.

RUSSELL, S. et NORVIG, P. (2010). *Intelligence artificielle, 3e édition*. Pearson Education, Paris.

SHEARER, R., MOTIK, B. et HORROCKS, I. (2008). HermiT : A Highly-Efficient OWL Reasoner.

TUTENEL, T., BIDARRA, R., SMELIK, R. M. et KRAKER, K. J. D. (2008). The role of semantics in games and simulations. *Computers in Entertainment CIE*, 6, 57.

VANACKEN, L., RAYMAEKERS, C. et CONINX, K. (2007). Introducing semantic information during conceptual modelling of interaction for virtual environments. *Proceedings of the 2007 workshop on Multimodal interfaces in semantic interaction*. 17–24.

ANNEXE A

SCHÉMA DE VALIDATION - FORMAT DE REPRÉSENTATION CONCEPTUELLE ABSTRAITE DE LA SCÈNE

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3              xmlns="">
4
5    <xsd:element name="corpus" type="corpusType"/>
6
7    <xsd:complexType name="corpusType">
8      <xsd:sequence>
9        <xsd:element name="text" type="xsd:string" minOccurs="0"/>
10       <xsd:element name="sceneDeclaration" type="sceneDeclarationType"/>
11     </xsd:sequence>
12   </xsd:complexType>
13
14   <xsd:complexType name="sceneDeclarationType">
15     <xsd:sequence>
16       <xsd:element name="configuration" type="configurationType" maxOccurs="unbounded"/>
17     </xsd:sequence>
18   </xsd:complexType>
19
20   <!-- Configuration types -->
21   <xsd:complexType name="configurationType">
22     <xsd:sequence>
23       <xsd:element name="process" type="xsd:anyURI"/>
24     </xsd:sequence>
25   </xsd:complexType>
26
27   <xsd:complexType name="SpatialLocating">
28     <xsd:complexContent>
29       <xsd:extension base="configurationType">
30         <xsd:sequence>
31           <xsd:element name="locatum" type="SimpleThing"/>
32           <xsd:choice>
33             <xsd:element name="location" type="GeneralizedLocation"/>
34             <xsd:element name="route" type="GeneralizedRoute"/>
35           </xsd:choice>
36         </xsd:sequence>
37       </xsd:extension>
38     </xsd:complexContent>
39   </xsd:complexType>
40
41   <xsd:complexType name="SpatialTemporalLocating">
42     <xsd:complexContent>
43       <xsd:extension base="SpatialLocating">
44         <xsd:choice>
45           <xsd:element name="dateTime" type="xsd:dateTime"/>
```

```
46            <xsd:element name="date" type="xsd:date"/>
47            <xsd:element name="time" type="xsd:time"/>
48          </xsd:choice>
49        </xsd:extension>
50      </xsd:complexContent>
51    </xsd:complexType>
52
53    <xsd:complexType name="NonAffectingAction">
54      <xsd:complexContent>
55        <xsd:extension base="configurationType">
56          <xsd:sequence>
57            <xsd:element name="actor" type="SimpleThing"/>
58          </xsd:sequence>
59        </xsd:extension>
60      </xsd:complexContent>
61    </xsd:complexType>
62
63    <xsd:complexType name="NonAffectingDirectedMotion">
64      <xsd:complexContent>
65        <xsd:extension base="NonAffectingAction">
66          <xsd:sequence>
67            <xsd:element name="placement" type="GeneralizedLocation" minOccurs="0"/>
68            <xsd:element name="motionDirection" type="GeneralizedLocation" minOccurs="0"/>
69            <xsd:element name="route" type="GeneralizedRoute" minOccurs="0"/>
70            <xsd:element name="direction" type="GeneralizedLocation" minOccurs="0"/>
71          </xsd:sequence>
72        </xsd:extension>
73      </xsd:complexContent>
74    </xsd:complexType>
75
76    <xsd:complexType name="NonAffectingOrientationChange">
77      <xsd:complexContent>
78        <xsd:extension base="NonAffectingAction">
79          <xsd:sequence>
80            <xsd:element name="route" type="GeneralizedRoute" minOccurs="0"/>
81            <xsd:choice>
82              <xsd:element name="orientationDirection" type="GeneralizedLocation"/>
83              <xsd:element name="orientationRoute" type="GeneralizedRoute"/>
84            </xsd:choice>
85          </xsd:sequence>
86        </xsd:extension>
87      </xsd:complexContent>
88    </xsd:complexType>
89
90    <xsd:complexType name="NonAffectingSimpleMotion">
91      <xsd:complexContent>
92        <xsd:extension base="NonAffectingAction">
93          <xsd:sequence>
94            <xsd:element name="placement" type="GeneralizedLocation" minOccurs="0"/>
95          </xsd:sequence>
96        </xsd:extension>
97      </xsd:complexContent>
98    </xsd:complexType>
99
100   <xsd:complexType name="AffectingAction">
```

```
101     <xsd:complexContent>
102       <xsd:extension base="configurationType">
103         <xsd:sequence>
104           <xsd:element name="actor" type="SimpleThing"/>
105           <xsd:element name="actee" type="SimpleThing"/>
106         </xsd:sequence>
107       </xsd:extension>
108     </xsd:complexContent>
109   </xsd:complexType>
110
111   <xsd:complexType name="AffectingDirectedMotion">
112     <xsd:complexContent>
113       <xsd:extension base="AffectingAction">
114         <xsd:sequence>
115           <xsd:element name="placement" type="GeneralizedLocation" minOccurs="0"/>
116           <xsd:element name="motionDirection" type="GeneralizedLocation" minOccurs="0"/>
117           <xsd:element name="route" type="GeneralizedRoute" minOccurs="0"/>
118           <xsd:element name="direction" type="GeneralizedLocation" minOccurs="0"/>
119         </xsd:sequence>
120       </xsd:extension>
121     </xsd:complexContent>
122   </xsd:complexType>
123
124   <xsd:complexType name="AffectingOrientationChange">
125     <xsd:complexContent>
126       <xsd:extension base="AffectingAction">
127         <xsd:sequence>
128           <xsd:element name="route" type="GeneralizedRoute" minOccurs="0"/>
129           <xsd:choice>
130             <xsd:element name="orientationDirection" type="GeneralizedLocation"/>
131             <xsd:element name="orientationRoute" type="GeneralizedRoute"/>
132           </xsd:choice>
133         </xsd:sequence>
134       </xsd:extension>
135     </xsd:complexContent>
136   </xsd:complexType>
137
138   <xsd:complexType name="AffectingSimpleMotion">
139     <xsd:complexContent>
140       <xsd:extension base="AffectingAction">
141         <xsd:sequence>
142           <xsd:element name="placement" type="GeneralizedLocation" minOccurs="0"/>
143         </xsd:sequence>
144       </xsd:extension>
145     </xsd:complexContent>
146   </xsd:complexType>
147
148   <!-- Route type -->
149   <xsd:complexType name ="GeneralizedRoute">
150     <xsd:sequence>
151       <xsd:element name="source" type="GeneralizedLocation" minOccurs="0"/>
152       <xsd:element name="pathPlacement" type="GeneralizedPathLocation" minOccurs="0"/>
153       <xsd:element name="destination" type="GeneralizedLocation" minOccurs="0"/>
154     </xsd:sequence>
155   </xsd:complexType>
```

```
156
157   <!-- Location types -->
158   <xsd:complexType name="GeneralizedLocation">
159     <xsd:sequence>
160       <xsd:element name="relatum" type="SimpleThing"/>
161       <xsd:element name="spatialModality" type="SpatialModality"/>
162     </xsd:sequence>
163   </xsd:complexType>
164
165   <xsd:complexType name="GeneralizedPathLocation">
166     <xsd:complexContent>
167       <xsd:extension base="GeneralizedLocation">
168         <xsd:sequence>
169           <xsd:element name="nextPathPlacement" type="GeneralizedPathLocation" minOccurs="0"/>
170           <xsd:element name="nextPathIndication" type="GeneralizedPathLocation" minOccurs="0"/>
171         </xsd:sequence>
172       </xsd:extension>
173     </xsd:complexContent>
174   </xsd:complexType>
175
176   <!-- Object types -->
177   <xsd:complexType name="SimpleThing">
178     <xsd:sequence>
179       <xsd:element name="name" type="xsd:string"/>
180       <xsd:element name="class" type="xsd:anyURI" maxOccurs="unbounded"/>
181     </xsd:sequence>
182   </xsd:complexType>
183
184   <!-- Spatial Modality -->
185   <xsd:complexType name="SpatialModality">
186     <xsd:sequence>
187       <xsd:element name="type" type="xsd:anyURI"/>
188     </xsd:sequence>
189   </xsd:complexType>
190
191 </xsd:schema>
```

ANNEXE B

SCHÉMA DE VALIDATION - FORMAT DE REPRÉSENTATION DE L'INFORMATION DE LA SCÈNE STATIQUE

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3              xmlns="">
4
5    <xsd:element name="scene" type="sceneType"/>
6
7    <xsd:complexType name="sceneType">
8      <xsd:sequence>
9        <xsd:element name="object" type="objectType" maxOccurs="unbounded"/>
10     </xsd:sequence>
11   </xsd:complexType>
12
13   <!-- Object types -->
14   <xsd:complexType name="objectType">
15     <xsd:sequence>
16       <xsd:element name="id" type="xsd:string"/>
17       <xsd:element name="class" type="xsd:anyURI"/>
18       <xsd:element name="controller" type="controllerType" minOccurs="0"/>
19     </xsd:sequence>
20   </xsd:complexType>
21
22   <!-- Controller types -->
23   <xsd:complexType name="controllerType">
24     <xsd:sequence>
25       <xsd:element name="id" type="xsd:string"/>
26       <xsd:element name="type" type="xsd:anyURI"/>
27     </xsd:sequence>
28   </xsd:complexType>
29
30 </xsd:schema>
```

ANNEXE C

LISTE DES ACTIONS DU DOMAINE

```
(:action take
  :parameters (
    ?human - Human
    ?object - Object
    ?support - Support
    )
  :precondition (and
    (at ?human ?object)
    (supporting ?support ?object)
    (not (action-mutex-enabled))
    )
  :effect (and
    (holding ?human ?object)
    (not (supporting ?support ?object))
    (has_taken ?human ?object)
    (has_taken_from ?human ?object ?support)
    )
  )

(:action take
  :parameters (
    ?human - Human
    ?object - Object
    ?container - Container
    )
  :precondition (and
    (at ?human ?object)
    (containing ?container ?object)
    (not (action-mutex-enabled))
    )
  :effect (and
    (holding ?human ?object)
    (not (containing ?container ?object))
    (has_taken ?human ?object)
    (has_taken_from ?human ?object ?container)
    )
  )

(:action put_on
  :parameters (
    ?human - Human
    ?object - Object
    ?support - Support
    )
  :precondition (and
    (at ?human ?support)
    (holding ?human ?object)
```

```
48      (not (action-mutex-enabled))
49      )
50    :effect (and
51      (supporting ?support ?object)
52      (not (holding ?human ?object))
53      (has_put ?human ?object)
54      (has_put_on ?human ?object ?support)
55      )
56    )
57
58  (:action place_in
59    :parameters (
60      ?human - Human
61      ?object - Object
62      ?container - Container
63      )
64    :precondition (and
65      (at ?human ?container)
66      (holding ?human ?object)
67      (not (action-mutex-enabled))
68      )
69    :effect (and
70      (containing ?container ?object)
71      (not (holding ?human ?object))
72      (has_put ?human ?object)
73      (has_placed_in ?human ?object ?container)
74      )
75    )
76
77  (:action get_in
78    :parameters (
79      ?human - Human
80      ?humanContainer - HumanContainer
81      ?ground - Ground
82      )
83    :precondition (and
84      (at ?human ?humanContainer)
85      (supporting ?ground ?human)
86      (not (action-mutex-enabled))
87      )
88    :effect (and
89      (containing ?humanContainer ?human)
90      (not (supporting ?ground ?human))
91      (has_gotten_in ?human ?humanContainer)
92      )
93    )
94
95  (:action get_out
96    :parameters (
97      ?human - Human
98      ?humanContainer - HumanContainer
99      ?ground - Ground
100     )
101   :precondition (and
102     (containing ?humanContainer ?human)
```

```
103       (not (action-mutex-enabled))
104       )
105     :effect (and
106       (supporting ?ground ?human)
107       (not (containing ?humanContainer ?human))
108       (has_gotten_out ?human ?humanContainer)
109       )
110     )
111
112   (:action get_on
113     :parameters (
114       ?human - Human
115       ?support - BelowHorizontalSupport
116       ?ground - Ground
117       )
118     :precondition (and
119       (at ?human ?support)
120       (supporting ?ground ?human)
121       (not (action-mutex-enabled))
122       )
123     :effect (and
124       (supporting ?support ?human)
125       (not (supporting ?ground ?human))
126       (has_gotten_on ?human ?support)
127       )
128     )
129
130   (:action get_off
131     :parameters (
132       ?human - Human
133       ?support - BelowHorizontalSupport
134       ?ground - Ground
135       )
136     :precondition (and
137       (supporting ?support ?human)
138       (not (action-mutex-enabled))
139       )
140     :effect (and
141       (supporting ?ground ?human)
142       (not (supporting ?support ?human))
143       (has_gotten_off ?human ?support)
144       )
145     )
146
147   (:action sit_down
148     :parameters (
149       ?human - Human
150       ?sittingFurniture - SittingFurniture
151       ?ground - Ground
152       )
153     :precondition (and
154       (at ?human ?sittingFurniture)
155       (supporting ?ground ?human)
156       (not (action-mutex-enabled))
157       )
```

```
158     :effect (and
159       (supporting ?sittingFurniture ?human)
160       (not (supporting ?ground ?human))
161       (has_sat_down ?human)
162       (has_sat_down_on ?human ?sittingFurniture)
163       (has_changed_posture ?human)
164       )
165     )
166
167   (:action lie_down
168     :parameters (
169       ?human - Human
170       ?sleepingFurniture - SleepingFurniture
171       ?ground - Ground
172       )
173     :precondition (and
174       (at ?human ?sleepingFurniture)
175       (supporting ?ground ?human)
176       (not (action-mutex-enabled))
177       )
178     :effect (and
179       (supporting ?sleepingFurniture ?human)
180       (not (supporting ?ground ?human))
181       (has_lied_down ?human)
182       (has_lied_down_on ?human ?sleepingFurniture)
183       (has_changed_posture ?human)
184       )
185     )
186
187   (:action stand_up
188     :parameters (
189       ?human - Human
190       ?furniture - Furniture
191       ?ground - Ground
192       )
193     :precondition (and
194       (supporting ?furniture ?human)
195       (not (action-mutex-enabled))
196       )
197     :effect (and
198       (supporting ?ground ?human)
199       (not (supporting ?furniture ?human))
200       (has_stood_up ?human)
201       (has_stood_up_from ?human ?furniture)
202       (has_changed_posture ?human)
203       )
204     )
205
206   (:action push
207     :parameters (
208       ?human - Human
209       ?pushedObject - Object
210       )
211     :precondition (and
212       (at ?human ?pushedObject)
```

```
213       (not (action-mutex-enabled))
214       )
215     :effect (
216       has_pushed ?human ?pushedObject
217       )
218     )
219
220   (:action push_to
221     :parameters (
222       ?human - Human
223       ?pushedObject - Object
224       ?targetObject - Object
225       )
226     :precondition (and
227       (at ?human ?pushedObject)
228       (not (action-mutex-enabled))
229       )
230     :effect (
231       has_pushed_to ?human ?pushedObject ?targetObject
232       )
233     )
234
235   (:action throw
236     :parameters (
237       ?human - Human
238       ?thrownObject - Object
239       )
240     :precondition (and
241       (holding ?human ?thrownObject)
242       (not (action-mutex-enabled))
243       )
244     :effect (and
245       (not (holding ?human ?thrownObject))
246       (has_thrown ?human ?thrownObject)
247       )
248     )
249
250   (:action throw_at
251     :parameters (
252       ?human - Human
253       ?thrownObject - Object
254       ?aimedObject - Object
255       )
256     :precondition (and
257       (is_aiming ?human ?thrownObject ?aimedObject)
258       (holding ?human ?thrownObject)
259       (action-mutex-enabled)
260       )
261     :effect (and
262       (not (is_aiming ?human ?thrownObject ?aimedObject))
263       (not (holding ?human ?thrownObject))
264       (has_thrown_at ?human ?thrownObject ?aimedObject)
265       (not (action-mutex-enabled))
266       )
267     )
```

```
268
269 (:action throw_at
270   :parameters (
271     ?human - Human
272     ?thrownObject - Object
273     ?nonObjectTarget - Support
274     )
275   :precondition (and
276     (is_aiming ?human ?thrownObject ?nonObjectTarget)
277     (holding ?human ?thrownObject)
278     (action-mutex-enabled)
279     )
280   :effect (and
281     (not (is_aiming ?human ?thrownObject ?nonObjectTarget))
282     (not (holding ?human ?thrownObject))
283     (has_thrown_at ?human ?thrownObject ?nonObjectTarget)
284     (not (action-mutex-enabled))
285     )
286   )
287
288 (:action throw_in
289   :parameters (
290     ?human - Human
291     ?thrownObject - Object
292     ?container - Container
293     )
294   :precondition (and
295     (is_aiming ?human ?thrownObject ?container)
296     (holding ?human ?thrownObject)
297     (action-mutex-enabled)
298     )
299   :effect (and
300     (not (is_aiming ?human ?thrownObject ?container))
301     (not (holding ?human ?thrownObject))
302     (has_thrown_in ?human ?thrownObject ?container)
303     (not (action-mutex-enabled))
304     )
305   )
306
307 (:action pour
308   :parameters (
309     ?human - Human
310     ?water - Water
311     ?waterSource - WaterSource
312     ?liquidContainer - LiquidContainer
313     )
314   :precondition (and
315     (at ?human ?waterSource)
316     (holding ?human ?liquidContainer)
317     (containing ?waterSource ?water)
318     (not (action-mutex-enabled))
319     )
320   :effect (and
321     (containing ?liquidContainer ?water)
322     (has_poured ?human ?water)
```

```
323       (has_poured_from ?human ?water ?waterSource)
324       (has_poured_in ?human ?water ?liquidContainer)
325       )
326     )
327
328   (:action drink-begin
329     :parameters (
330       ?human - Human
331       ?drinkableLiquid - DrinkableLiquid
332       ?glassware - Glassware
333       )
334     :precondition (and
335       (holding ?human ?glassware)
336       (containing ?glassware ?drinkableLiquid)
337       (not (action-mutex-enabled))
338       )
339     :effect (and
340       (is_drinking ?human)
341       (is_drinking ?human ?drinkableLiquid)
342       (action-mutex-enabled)
343       )
344     )
345
346   (:action drink-end
347     :parameters (
348       ?human - Human
349       ?drinkableLiquid - DrinkableLiquid
350       ?glassware - Glassware
351       )
352     :precondition (and
353       (is_drinking ?human)
354       (is_drinking ?human ?drinkableLiquid)
355       (action-mutex-enabled)
356       )
357     :effect (and
358       (not (containing ?glassware ?drinkableLiquid))
359       (not (is_drinking ?human))
360       (not (is_drinking ?human ?drinkableLiquid))
361       (has_drunk ?human ?drinkableLiquid)
362       (has_ingested ?human)
363       (not (action-mutex-enabled))
364       )
365     )
366
367   (:action eat-begin
368     :parameters (
369       ?human - Human
370       ?food - EatableWithoutUtensilFood
371       )
372     :precondition (and
373       (holding ?human ?food)
374       (not (action-mutex-enabled))
375       )
376     :effect (and
377       (is_eating ?human)
```

```
378       (is_eating ?human ?food)
379       (action-mutex-enabled)
380       )
381     )
382
383   (:action eat-begin
384     :parameters (
385       ?human - Human
386       ?food - EatableWithForkFood
387       ?fork - Fork
388       )
389     :precondition (and
390       (at ?human ?food)
391       (holding ?human ?fork)
392       (not (action-mutex-enabled))
393       )
394     :effect (and
395       (is_eating ?human)
396       (is_eating ?human ?food)
397       (action-mutex-enabled)
398       )
399     )
400
401   (:action eat-begin
402     :parameters (
403       ?human - Human
404       ?food - EatableWithSpoonFood
405       ?spoon - Spoon
406       )
407     :precondition (and
408       (at ?human ?food)
409       (holding ?human ?spoon)
410       (not (action-mutex-enabled))
411       )
412     :effect (and
413       (is_eating ?human)
414       (is_eating ?human ?food)
415       (action-mutex-enabled)
416       )
417     )
418
419   (:action eat-begin
420     :parameters (
421       ?human - Human
422       ?food - EatableWithForkFood
423       ?fork - Fork
424       ?tableware - Tableware
425       )
426     :precondition (and
427       (containing ?tableware ?food)
428       (holding ?human ?tableware)
429       (holding ?human ?fork)
430       (not (action-mutex-enabled))
431       )
432     :effect (and
```

```
433       (is_eating ?human)
434       (is_eating ?human ?food)
435       (action-mutex-enabled)
436       )
437     )
438
439   (:action eat-begin
440     :parameters (
441       ?human - Human
442       ?food - EatableWithSpoonFood
443       ?spoon - Spoon
444       ?tableware - Tableware
445       )
446     :precondition (and
447       (containing ?tableware ?food)
448       (holding ?human ?tableware)
449       (holding ?human ?spoon)
450       (not (action-mutex-enabled))
451       )
452     :effect (and
453       (is_eating ?human)
454       (is_eating ?human ?food)
455       (action-mutex-enabled)
456       )
457     )
458
459   (:action eat-end
460     :parameters (
461       ?human - Human
462       ?food - Food
463       )
464     :precondition (and
465       (is_eating ?human)
466       (is_eating ?human ?food)
467       (action-mutex-enabled)
468       )
469     :effect (and
470       (not (is_eating ?human))
471       (not (is_eating ?human ?food))
472       (has_eaten ?human ?food)
473       (has_ingested ?human)
474       (not (action-mutex-enabled))
475       )
476     )
477
478   (:action wash-itself-begin
479     :parameters (
480       ?human - Human
481       ?bathingFixture - BathingFixture
482       )
483     :precondition (and
484       (containing ?bathingFixture ?human)
485       (not (action-mutex-enabled))
486       )
487     :effect (and
```

```
488       (is_washing_itself ?human)
489       (action-mutex-enabled)
490       )
491     )
492
493   (:action wash-itself-end
494     :parameters (
495       ?human - Human
496       ?bathingFixture - BathingFixture
497       )
498     :precondition (and
499       (is_washing_itself ?human)
500       (containing ?bathingFixture ?human)
501       (action-mutex-enabled)
502       )
503     :effect (and
504       (not (is_washing_itself ?human))
505       (has_washed_itself ?human)
506       (not (action-mutex-enabled))
507       )
508     )
509
510   (:action wash-begin
511     :parameters (
512       ?human - Human
513       ?dish - Dish
514       ?sink - Sink
515       )
516     :precondition (and
517       (holding ?human ?dish)
518       (at ?human ?sink)
519       (not (action-mutex-enabled))
520       )
521     :effect (and
522       (is_washing ?human ?dish)
523       (action-mutex-enabled)
524       )
525     )
526
527   (:action wash-begin
528     :parameters (
529       ?human - Human
530       ?clothing - Clothing
531       ?washer - Washer
532       )
533     :precondition (and
534       (containing ?washer ?clothing)
535       (at ?human ?washer)
536       (not (action-mutex-enabled))
537       )
538     :effect (and
539       (is_washing ?human ?clothing)
540       (action-mutex-enabled)
541       )
542     )
```

```
543
544 (:action wash-end
545   :parameters (
546     ?human - Human
547     ?object - Object
548     )
549   :precondition (and
550     (is_washing ?human ?object)
551     (action-mutex-enabled)
552     )
553   :effect (and
554     (not (is_washing ?human ?object))
555     (has_washed ?human ?object)
556     (not (action-mutex-enabled))
557     )
558   )
559
560 (:action aim
561   :parameters (
562     ?human - Human
563     ?aimingObject - Object
564     ?aimedObject - Object
565     )
566   :precondition (and
567     (holding ?human ?aimingObject)
568     (not (action-mutex-enabled))
569     )
570   :effect (and
571     (is_aiming ?human ?aimingObject ?aimedObject)
572     (action-mutex-enabled)
573     )
574   )
575
576 (:action aim
577   :parameters (
578     ?human - Human
579     ?aimingObject - Object
580     ?nonObjectTarget - Support
581     )
582   :precondition (and
583     (holding ?human ?aimingObject)
584     (not (action-mutex-enabled))
585     )
586   :effect (and
587     (is_aiming ?human ?aimingObject ?nonObjectTarget)
588     (action-mutex-enabled)
589     )
590   )
591
592 (:action aim-end
593   :parameters (
594     ?human - Human
595     ?aimingObject - Object
596     ?aimedObject - Object
597     )
```

```
598     :precondition (and
599       (is_aiming ?human ?aimingObject ?aimedObject)
600       (holding ?human ?aimingObject)
601       (action-mutex-enabled)
602       )
603     :effect (and
604       (not (is_aiming ?human ?aimingObject ?aimedObject))
605       (has_aimed ?human)
606       (has_aimed_at ?human ?aimingObject ?aimedObject)
607       (not (action-mutex-enabled))
608       )
609     )
610
611   (:action move-begin
612     :parameters (
613       ?human - Human
614       ?ground - Ground
615       )
616     :precondition (and
617       (supporting ?ground ?human)
618       (not (action-mutex-enabled))
619       )
620     :effect (and
621       (is_moving ?human)
622       (action-mutex-enabled)
623       )
624     )
625
626   (:action move-end
627     :parameters (
628       ?human - Human
629       ?ground - Ground
630       )
631     :precondition (and
632       (is_moving ?human)
633       (supporting ?ground ?human)
634       (action-mutex-enabled)
635       )
636     :effect (and
637       (not (is-moving ?human))
638       (has_moved ?human)
639       (not (action-mutex-enabled))
640       )
641     )
642
643   (:action move_to
644     :parameters (
645       ?human - Human
646       ?initialObject - Object
647       ?finalObject - Object
648       )
649     :precondition (and
650       (at ?human ?initialObject)
651       (is_moving ?human)
652       )
```

```
653     :effect (and
654       (not (at ?human ?initialObject))
655       (at ?human ?finalObject)
656       (has_moved_from ?human ?initialObject)
657       (has_moved_to ?human ?finalObject)
658       )
659     )
660
661   (:action move_to
662     :parameters (
663       ?human - Human
664       ?object - Object
665     )
666     :precondition (and
667       (at_start ?human)
668       (is_moving ?human)
669       )
670     :effect (and
671       (not (at_start ?human))
672       (at ?human ?object)
673       (has_moved_to ?human ?object)
674       )
675     )
```